# proyecto_elvis_2_4

November 3, 2024

## 1 Análisis del Conjunto de Datos de Precios de Aguacate

**Conjunto de Datos de Precios de Aguacate**: El conjunto de datos "Precios de Aguacate", obtenido de Kaggle, es un conjunto de datos ampliamente utilizado para proyectos de análisis de datos y aprendizaje automático. Proporciona datos históricos sobre precios y ventas de aguacates en varias regiones de los Estados Unidos. Este conjunto de datos es valioso para entender las tendencias en los precios de los aguacates, los volúmenes de ventas y su relación con diferentes factores.

### 1.1 Atributos Clave

- **Columnas**: El conjunto de datos incluye varias columnas de información. Algunas de las columnas clave típicamente encontradas en este conjunto de datos incluyen:
  - **Fecha** (`Date`): La fecha de observación.
  - **Precio Promedio** (`AveragePrice`): El precio promedio de los aguacates.
  - **Volumen Total** (`Total Volume`): El volumen total de aguacates vendidos.
  - **4046**: Volumen de aguacates Hass pequeños vendidos.
  - **4225**: Volumen de aguacates Hass grandes vendidos.
  - **4770**: Volumen de aguacates Hass extra grandes vendidos.
  - **Bolsas Totales** (`Total Bags`): Total de bolsas de aguacates vendidas.
  - **Bolsas Pequeñas** (`Small Bags`): Bolsas de aguacates pequeños vendidas.
  - **Bolsas Grandes** (`Large Bags`): Bolsas de aguacates grandes vendidas.
  - **Bolsas Extra Grandes** (`XLarge Bags`): Bolsas de aguacates extra grandes vendidas.
  - **Tipo** (`Type`): El tipo de aguacates, generalmente categorizados como convencionales u orgánicos.
  - **Región** (`Region`): La región o ciudad dentro de los Estados Unidos donde se registraron los datos.
- **Rango de Fechas**: El conjunto de datos abarca un rango de fechas, lo que permite el análisis de series de tiempo. Puedes examinar cómo cambian los precios y ventas de aguacates a lo largo de diferentes estaciones y años.
- **Regiones**: Se proporciona información para varias regiones o ciudades a través de los Estados Unidos, lo que permite el análisis de variaciones de precios y ventas en diferentes mercados.
- **Tipos**: El conjunto de datos distingue entre diferentes tipos de aguacates, como convencionales y orgánicos, lo que puede ser útil para comparar tendencias de precios entre estas categorías.

- **Volumen**: Están disponibles datos sobre el volumen total de aguacates vendidos. Esta métrica de volumen se utiliza a menudo para analizar la demanda del mercado.
- **Precio Promedio**: El conjunto de datos contiene el precio promedio de los aguacates, una métrica fundamental para entender las tendencias de precios.

## 1.2 Casos de Uso

- Este conjunto de datos se utiliza comúnmente para aprender y practicar el análisis de datos, visualización de datos y modelado de regresión en proyectos de ciencia de datos y aprendizaje automático.
- Sirve como un recurso valioso para entender cómo trabajar con datos del mundo real, extraer conocimientos y tomar decisiones basadas en datos.

---

## 1.3 Actividades de Análisis

### 1.3.1 1. Análisis de Series Temporales

**Resumen:** El análisis de series temporales permite identificar patrones, tendencias y estacionalidades en los precios y volúmenes de ventas de aguacates a lo largo del tiempo.

1. **Descomposición de Series Temporales de Precios:**
   - **Uso de Datos:** Usa la columna `AveragePrice` y `Date`.
   - **Esperado:** Utiliza la función `seasonal_decompose` de la librería `statsmodels` para descomponer la serie temporal de precios en componentes de tendencia, estacionalidad y ruido.
     - Convierte `Date` a tipo datetime usando `pd.to_datetime()`.
     - Agrupa los datos por `Date` y calcula el promedio de `AveragePrice` utilizando `groupby()` si es necesario.
     - Visualiza los componentes descompuestos usando `matplotlib` para cada uno de ellos.
2. **Análisis de Estacionalidad por Región:**
   - **Uso de Datos:** Usa las columnas `AveragePrice`, `Date` y `Total Volume`.
   - **Esperado:** Utiliza gráficos de líneas para visualizar cómo varían los precios de aguacates por región a lo largo de diferentes estaciones del año.
     - Agrupa los datos por `region` y `Date` utilizando `groupby()`.
     - Calcula el promedio de `AveragePrice` para cada región.
     - Representa gráficamente las tendencias utilizando `plt.plot()` de `matplotlib`.
3. **Comparación de Precios Promedio Mensuales:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `Date`.
   - **Esperado:** Calcula y compara los precios promedio mensuales.
     - Agrupa los datos por mes usando `pd.Grouper` con `freq='M'`.
     - Calcula el promedio de `AveragePrice` para cada mes con `mean()`.
     - Visualiza los resultados con un gráfico de líneas usando `plt.plot()`.
4. **Tendencia de Ventas a lo Largo del Tiempo:**
   - **Uso de Datos:** Usa las columnas `Total Volume` y `Date`.
   - **Esperado:** Analiza cómo varía el volumen total de ventas a lo largo del tiempo.
     - Agrupa los datos por `Date` y suma el `Total Volume` usando `groupby()`.

- Visualiza los resultados usando un gráfico de líneas con `plt.plot()` para mostrar la tendencia.

5. **Análisis de Cambios en Precios Anuales:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `year`.
   - **Esperado:** Observa las diferencias anuales en los precios promedio.
     - Agrupa los datos por `year` utilizando `groupby()`.
     - Calcula el promedio de `AveragePrice` para cada año.
     - Representa los resultados en un gráfico de barras usando `plt.bar()` que compare los precios de cada año.

### 1.3.2 2. Gráficos para Visualización de Datos

**Resumen:** La visualización de datos es clave para identificar patrones y relaciones entre diferentes variables. Los gráficos apropiados pueden proporcionar información valiosa sobre el comportamiento de los precios y volúmenes de ventas.

1. **Gráfico de Violín de Volumen de Ventas por Región:**
   - **Uso de Datos:** Usa las columnas `Total Volume` y `region`.
   - **Esperado:** Visualiza la distribución de ventas en diferentes regiones.
     - Utiliza la función `violinplot` de `seaborn` para crear gráficos de violín.
     - Configura los ejes para mostrar la relación entre `Total Volume` y `region`.
     - Añade etiquetas y títulos usando `plt.title()` y `plt.xlabel()` para facilitar la interpretación.
2. **Boxplot Comparativo de Precios entre Años:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `year`.
   - **Esperado:** Genera boxplots para comparar la distribución de precios.
     - Utiliza `boxplot` de `seaborn` para crear boxplots que comparen `AveragePrice` entre diferentes años.
     - Asegúrate de que cada boxplot represente un año diferente.
     - Incluye etiquetas y títulos descriptivos usando `plt.title()`.
3. **Histograma de Volumen Total de Ventas:**
   - **Uso de Datos:** Usa la columna `Total Volume`.
   - **Esperado:** Crea un histograma para mostrar la distribución del volumen total de ventas.
     - Utiliza `hist()` de `matplotlib` para crear el histograma.
     - Ajusta el número de bins para una visualización clara usando el parámetro `bins`.
     - Añade etiquetas y un título que describa lo que se muestra.
4. **Gráfico de Barras de Ventas por Tipo de Bolsa:**
   - **Uso de Datos:** Utiliza las columnas `Total Bags`, `Small Bags`, `Large Bags` y `XLarge Bags`.
   - **Esperado:** Compara las ventas de diferentes tipos de bolsas.
     - Suma los volúmenes de ventas por tipo de bolsa utilizando `sum()`.
     - Crea un gráfico de barras con `plt.bar()` para mostrar las diferencias en ventas.
     - Asegúrate de incluir etiquetas para cada tipo de bolsa.
5. **Gráfico de Líneas de Precios Promedios por Año:**
   - **Uso de Datos:** Utiliza las columnas `AveragePrice` y `year`.
   - **Esperado:** Visualiza la tendencia de precios promedio a lo largo de los años.
     - Agrupa los datos por `year` y calcula el promedio de `AveragePrice`.
     - Usa `plt.plot()` para crear un gráfico de líneas que muestre la evolución de precios.

- Añade un título y etiquetas descriptivas a los ejes usando `plt.title()` y `plt.xlabel()`.

#### 1.3.3 3. Elasticidad del Precio

**Resumen:** El análisis de elasticidad precio-demanda permite evaluar cómo los cambios en los precios afectan la demanda de aguacates. Comprender la elasticidad puede ayudar a formular estrategias de precios más efectivas.

La fórmula de elasticidad precio-demanda es:

$$E_d = \frac{\%\text{Cambio en la cantidad demandada}}{\%\text{Cambio en el precio}} = \frac{\Delta Q/Q}{\Delta P/P}$$

1. **Elasticidad Precio-Demanda por Año:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `Total Volume`.
   - **Esperado:** Calcula la elasticidad del precio de la demanda para cada año.
     - Calcula la variación porcentual de `Total Volume` y `AveragePrice` utilizando `pd.pct_change()`.
     - Utiliza la fórmula de elasticidad para determinar la sensibilidad de la demanda respecto al precio.
     - Presenta los resultados en un gráfico de líneas usando `plt.plot()` para mostrar la elasticidad por año.
2. **Comparación de Elasticidad en Diferentes Mercados:**
   - **Uso de Datos:** Utiliza las columnas `Total Volume` y `AveragePrice`.
   - **Esperado:** Calcula la elasticidad del precio de la demanda en diferentes regiones.
     - Agrupa los datos por `region` y calcula la elasticidad para cada región utilizando `pd.pct_change()`.
     - Presenta un gráfico de barras que muestre la elasticidad por región usando `plt.bar()`.
3. **Elasticidad a Nivel de Tipo de Bolsa:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `Total Bags`.
   - **Esperado:** Calcula la elasticidad del precio de la demanda específica para cada tipo de bolsa.
     - Suma los volúmenes de ventas por tipo de bolsa utilizando `groupby()` y `sum()`.
     - Calcula la elasticidad para cada tipo y presenta los resultados en un gráfico comparativo usando `plt.bar()`.
4. **Análisis de Elasticidad Comparativa entre Orgánicos y Convencionales:**
   - **Uso de Datos:** Usa las columnas `AveragePrice`, `Total Volume` y `type`.
   - **Esperado:** Compara la elasticidad de la demanda entre aguacates orgánicos y convencionales.
     - Agrupa los datos por `type` y calcula la elasticidad utilizando `pd.pct_change()`.
     - Presenta un gráfico que muestre la diferencia en elasticidad entre los dos tipos usando `plt.bar()`.
5. **Análisis de la Elasticidad Precios-Ventas:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `Total Volume`.
   - **Esperado:** Examina cómo las variaciones en `AveragePrice` afectan a `Total Volume`.
     - Realiza un análisis de la relación entre estas dos variables calculando la elasticidad.

- Presenta un gráfico de dispersión que muestre la relación y discute la tendencia observada utilizando `plt.scatter()` y `plt.plot()`.

#### 1.3.4 4. Análisis de Cohortes

**Resumen:** El análisis de cohortes permite agrupar datos según características específicas y observar cómo se comportan a lo largo del tiempo. Se centra en cohortes de precios y ventas para entender las dinámicas del mercado.

1. **Cohortes Basadas en Precios Promedios Trimestrales:**
   - **Uso de Datos:** Usa las columnas `AveragePrice`, `Total Volume` y `Date`.
   - **Esperado:** Crea cohortes trimestrales y analiza cambios en precios y volúmenes.
     - Agrupa los datos por trimestre usando `pd.Grouper` con `freq='Q'`.
     - Calcula el promedio de `AveragePrice` y suma `Total Volume` para cada cohorte.
     - Visualiza los resultados en un gráfico de líneas que muestre la evolución de las cohortes.
2. **Cohortes por Región y Fecha:**
   - **Uso de Datos:** Utiliza las columnas `AveragePrice`, `Total Volume`, `region` y `Date`.
   - **Esperado:** Analiza cómo varían las cohortes de diferentes regiones.
     - Agrupa los datos por `region` y `Date` usando `groupby()`.
     - Calcula el promedio de precios y volumen para cada cohorte.
     - Presenta los resultados en gráficos de barras que muestren comparaciones entre regiones.
3. **Análisis de Cohortes en Función del Tipo de Bolsa:**
   - **Uso de Datos:** Usa las columnas `Total Bags`, `Small Bags`, `Large Bags`, `XLarge Bags` y `Date`.
   - **Esperado:** Examina cómo se comportan las diferentes cohortes según el tipo de bolsa.
     - Agrupa los datos por tipo de bolsa y `Date`.
     - Calcula el volumen de ventas total y muestra los resultados en un gráfico de líneas.
4. **Cohortes de Clientes Basadas en Ventas:**
   - **Uso de Datos:** Usa las columnas `Total Volume`, `Date` y `region`.
   - **Esperado:** Analiza el comportamiento de las cohortes según el volumen de ventas.
     - Clasifica los clientes según su volumen de compras.
     - Visualiza las cohortes en gráficos de líneas o barras que muestren el comportamiento de compra a lo largo del tiempo.
5. **Evaluación de Retención de Ventas por Cohorte:**
   - **Uso de Datos:** Usa las columnas `Total Volume` y `Date`.
   - **Esperado:** Estudia cómo se retienen las ventas en cohortes a lo largo de un año.
     - Agrupa los datos por mes y cohortes.
     - Calcula la retención de ventas y visualiza los resultados en un gráfico de líneas que muestre las tasas de retención.

#### 1.3.5 5. Análisis de Correlación y Regresión

**Resumen:** Se centra en la identificación de relaciones significativas entre las variables numéricas y el desarrollo de modelos de regresión para hacer predicciones basadas en esas relaciones.

1. **Matriz de Correlación:**

- **Uso de Datos:** Utiliza las columnas numéricas del DataFrame (p. ej., `AveragePrice`, `Total Volume`, `4046`, `4225`, `4770`, `Total Bags`).
- **Esperado:**
  - Importa las librerías necesarias: `import seaborn as sns` y `import matplotlib.pyplot as plt`.
  - Calcula la matriz de correlación usando el método `.corr()` del DataFrame.
  - Visualiza la matriz utilizando `sns.heatmap()`.
  - Anota las correlaciones más significativas y discute su posible impacto en el análisis.

2. **Análisis de Dispersión entre Variables Clave:**
   - **Uso de Datos:** Selecciona variables numéricas de interés como `AveragePrice` y `Total Volume`.
   - **Esperado:**
     - Importa las librerías necesarias: `import seaborn as sns` y `import matplotlib.pyplot as plt`.
     - Crea un gráfico de dispersión con `sns.scatterplot()` para visualizar la relación entre `AveragePrice` y `Total Volume`.
     - Añade una línea de regresión utilizando `sns.regplot()` para ilustrar las tendencias.
     - Compara el ajuste de una regresión lineal frente a una polinómica.
3. **Predicciones Mensuales Usando Datos Trimestrales:**
   - **Uso de Datos:** Agrupa datos por trimestres y segmenta en meses utilizando `Date`, `AveragePrice`, y `Total Volume`.
   - **Esperado:**
     - Convierte la columna `Date` a tipo datetime si es necesario.
     - Agrupa los datos por trimestre y calcula el promedio de `AveragePrice` y `Total Volume`.
     - Utiliza los datos de los primeros 2 meses de un trimestre para predecir el precio del tercer mes.
     - Compara los resultados de las predicciones con los precios reales.
     - Evalúa la precisión de tus predicciones utilizando métricas como $R^2$ y RMSE.
4. **Predicciones Trimestrales:**
   - **Uso de Datos:** Agrupa los datos en trimestres usando solo variables numéricas.
   - **Esperado:**
     - Agrupa los datos por trimestres usando `pd.Grouper()` con `freq='Q'` para obtener promedios.
     - Usa los datos de 1 o 2 trimestres anteriores para predecir el siguiente trimestre ajustando modelos de regresión lineal y polinómica.
     - Compara los resultados de las predicciones con los precios reales.
     - Evalúa la precisión de tus predicciones utilizando métricas como $R^2$ y RMSE.
5. **Predicciones Anuales:**
   - **Uso de Datos:** Agrupa los datos en años, utilizando únicamente columnas numéricas.
   - **Esperado:**
     - Agrupa los datos por año utilizando `pd.Grouper()` con `freq='Y'`.
     - Usa los datos de 1 o 2 años anteriores para predecir el siguiente año ajustando modelos de regresión lineal y polinómica.
     - Evalúa la precisión de tus predicciones utilizando métricas como $R^2$ y RMSE.
6. **Desarrollo de Modelos de Regresión Múltiple:**
   - **Uso de Datos:** Selecciona varias variables numéricas como `Total Volume`, `4046`, `4225`, `4770`, y `Total Bags` para predecir `AveragePrice`.

- **Esperado:**
  - Define las variables independientes (X) y dependientes (y).
  - Ajusta modelos de regresión múltiple.
  - Compara su rendimiento utilizando métricas como $R^2$ y RMSE y discute las implicaciones de los resultados.

7. **Análisis de Coeficientes de Regresión Múltiple:**
   - **Uso de Datos:** Examina los coeficientes de los modelos de regresión múltiple ajustados.
   - **Esperado:**
     - Extrae los coeficientes del modelo ajustado.
     - Interpreta los coeficientes para entender el impacto de cada variable numérica en `AveragePrice`.
     - Comenta sobre las variables más significativas y su relevancia.
8. **Modelos de Regresión para Diferenciar Volúmenes de Ventas:**
   - **Uso de Datos:** Usa `AveragePrice`, `Total Volume`, `4046`, `4225`, y `4770`.
   - **Esperado:**
     - Ajusta modelos de regresión para analizar cómo los diferentes volúmenes de ventas afectan `AveragePrice`.
     - Compara los resultados de regresión lineal y polinómica.
     - Presenta las conclusiones de tus análisis.
9. **Análisis de la Influencia de las Ventas Totales en el Precio Promedio:**
   - **Uso de Datos:** Usa `Total Volume`, `AveragePrice`, y `Total Bags`.
   - **Esperado:**
     - Ajusta un modelo de regresión lineal y polinómica para ver cómo varía `AveragePrice` en función del volumen total de ventas.
     - Evalúa la significancia de los coeficientes y discute su relevancia.
10. **Regresión para Predecir el Precio Promedio Según el Volumen de Aguacates por Tipo:**
    - **Uso de Datos:** Usa `AveragePrice`, `4046`, `4225`, `4770`, y `Total Volume`.
    - **Esperado:**
      - Ajusta modelos de regresión lineal y polinómica.
      - Evalúa la efectividad de ambos modelos utilizando métricas como $R^2$ y RMSE.
      - Discute cuál modelo ofrece mejores predicciones y por qué, basándote en los resultados obtenidos.

Elvis

### 1.3.6 2. Gráficos para Visualización de Datos

**Resumen:** La visualización de datos es clave para identificar patrones y relaciones entre diferentes variables. Los gráficos apropiados pueden proporcionar información valiosa sobre el comportamiento de los precios y volúmenes de ventas.

1. **Gráfico de Violín de Volumen de Ventas por Región:**
   - **Uso de Datos:** Usa las columnas `Total Volume` y `region`.
   - **Esperado:** Visualiza la distribución de ventas en diferentes regiones.
     - Utiliza la función `violinplot` de `seaborn` para crear gráficos de violín.
     - Configura los ejes para mostrar la relación entre `Total Volume` y `region`.
     - Añade etiquetas y títulos usando `plt.title()` y `plt.xlabel()` para facilitar la interpretación.

[2]:
```
pip install matplotlib
```

```
Collecting matplotlib
  Downloading matplotlib-3.9.2-cp311-cp311-win_amd64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.0-cp311-cp311-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.54.1-cp311-cp311-win_amd64.whl.metadata (167 kB)
     ---------------------------------------- 0.0/167.0 kB ? eta -:--:--
     ------- -------------------------------- 30.7/167.0 kB 1.3 MB/s eta 0:00:01
     ---------------------------------------- 167.0/167.0 kB 3.3 MB/s eta 0:00:00
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.7-cp311-cp311-win_amd64.whl.metadata (6.4 kB)
Collecting numpy>=1.23 (from matplotlib)
  Downloading numpy-2.1.2-cp311-cp311-win_amd64.whl.metadata (59 kB)
     ---------------------------------------- 0.0/59.7 kB ? eta -:--:--
     ---------------------------------------- 59.7/59.7 kB ? eta 0:00:00
Requirement already satisfied: packaging>=20.0 in c:\users\egortega\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib) (24.1)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.0.0-cp311-cp311-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Using cached pyparsing-3.2.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\egortega\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\egortega\appdata\local\packa
ges\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from python-dateutil>=2.7->matplotlib)
(1.16.0)
Downloading matplotlib-3.9.2-cp311-cp311-win_amd64.whl (7.8 MB)
   ---------------------------------------- 0.0/7.8 MB ? eta -:--:--
   ---- ----------------------------------- 0.8/7.8 MB 26.4 MB/s eta 0:00:01
   -------- ------------------------------- 1.7/7.8 MB 18.4 MB/s eta 0:00:01
   ------------- -------------------------- 2.8/7.8 MB 22.6 MB/s eta 0:00:01
   --------------------- ------------------ 4.2/7.8 MB 22.4 MB/s eta 0:00:01
   ----------------------------- ---------- 5.7/7.8 MB 24.3 MB/s eta 0:00:01
   ---------------------------------- ----- 6.7/7.8 MB 27.3 MB/s eta 0:00:01
   ---------------------------------------  7.8/7.8 MB 26.3 MB/s eta 0:00:01
   ---------------------------------------  7.8/7.8 MB 26.3 MB/s eta 0:00:01
   ---------------------------------------  7.8/7.8 MB 26.3 MB/s eta 0:00:01
   ---------------------------------------  7.8/7.8 MB 26.3 MB/s eta 0:00:01
   ---------------------------------------- 7.8/7.8 MB 16.7 MB/s eta 0:00:00
Downloading contourpy-1.3.0-cp311-cp311-win_amd64.whl (217 kB)
   ---------------------------------------- 0.0/217.2 kB ? eta -:--:--
```

```
   ---------------------------------------- 217.2/217.2 kB 12.9 MB/s eta 0:00:00
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.54.1-cp311-cp311-win_amd64.whl (2.2 MB)
   ---------------------------------------- 0.0/2.2 MB ? eta -:--:--
   ---------------------------------- ----- 1.9/2.2 MB 41.2 MB/s eta 0:00:01
   ---------------------------------------  2.2/2.2 MB 34.9 MB/s eta 0:00:01
   ---------------------------------------- 2.2/2.2 MB 20.0 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp311-cp311-win_amd64.whl (56 kB)
   ---------------------------------------- 0.0/56.0 kB ? eta -:--:--
   ---------------------------------------- 56.0/56.0 kB 2.9 MB/s eta 0:00:00
Downloading numpy-2.1.2-cp311-cp311-win_amd64.whl (12.9 MB)
   ---------------------------------------- 0.0/12.9 MB ? eta -:--:--
   ---- ----------------------------------- 1.5/12.9 MB 31.8 MB/s eta 0:00:01
   -------- ------------------------------- 2.9/12.9 MB 31.0 MB/s eta 0:00:01
   ------------ --------------------------- 4.2/12.9 MB 29.6 MB/s eta 0:00:01
   ---------------- ----------------------- 5.4/12.9 MB 28.8 MB/s eta 0:00:01
   -------------------- ------------------- 6.7/12.9 MB 28.4 MB/s eta 0:00:01
   ------------------------ --------------- 7.9/12.9 MB 27.8 MB/s eta 0:00:01
   ---------------------------- ----------- 9.3/12.9 MB 28.1 MB/s eta 0:00:01
   -------------------------------- ------- 10.5/12.9 MB 28.5 MB/s eta 0:00:01
   ---------------------------------- ----- 12.1/12.9 MB 28.5 MB/s eta 0:00:01
   ---------------------------------------  12.9/12.9 MB 28.5 MB/s eta 0:00:01
   ---------------------------------------- 12.9/12.9 MB 25.2 MB/s eta 0:00:00
Downloading pillow-11.0.0-cp311-cp311-win_amd64.whl (2.6 MB)
   ---------------------------------------- 0.0/2.6 MB ? eta -:--:--
   --------------- ------------------------ 1.0/2.6 MB 59.5 MB/s eta 0:00:01
   ---------------------------------------- 2.6/2.6 MB 32.8 MB/s eta 0:00:00
Using cached pyparsing-3.2.0-py3-none-any.whl (106 kB)
Installing collected packages: pyparsing, pillow, numpy, kiwisolver, fonttools,
cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.0 cycler-0.12.1 fonttools-4.54.1
kiwisolver-1.4.7 matplotlib-3.9.2 numpy-2.1.2 pillow-11.0.0 pyparsing-3.2.0
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: C:\Users\egortega\AppData\Local\Microsoft\WindowsApps\P
ythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install
--upgrade pip
```

[5]:
```
pip install seaborn
```

```
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\egortega\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from seaborn) (2.1.2)
Collecting pandas>=1.2 (from seaborn)
  Downloading pandas-2.2.3-cp311-cp311-win_amd64.whl.metadata (19 kB)
```

```
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\egortega\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\egortega\appdata\loc
al\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\egortega\appdata\local\p
ackages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\egortega\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\egortega\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\egortega\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\egortega\appdata\local\pack
ages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\egortega\appdata\loc
al\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\egortega\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(2.9.0.post0)
Collecting pytz>=2020.1 (from pandas>=1.2->seaborn)
  Using cached pytz-2024.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas>=1.2->seaborn)
  Using cached tzdata-2024.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in c:\users\egortega\appdata\local\packa
ges\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Downloading pandas-2.2.3-cp311-cp311-win_amd64.whl (11.6 MB)
   ---------------------------------------- 0.0/11.6 MB ? eta -:--:--
   ---------------------------------------- 0.1/11.6 MB 1.7 MB/s eta 0:00:07
   -- ------------------------------------- 0.6/11.6 MB 7.5 MB/s eta 0:00:02
   ----- ---------------------------------- 1.6/11.6 MB 11.0 MB/s eta 0:00:01
   -------- ------------------------------- 2.6/11.6 MB 13.6 MB/s eta 0:00:01
   ------------ --------------------------- 3.8/11.6 MB 16.0 MB/s eta 0:00:01
```

```
   ---------------- ---------------------- 5.1/11.6 MB 18.1 MB/s eta 0:00:01
   ---------------------- ---------------- 6.8/11.6 MB 20.7 MB/s eta 0:00:01
   --------------------------- ----------- 8.2/11.6 MB 21.9 MB/s eta 0:00:01
   ------------------------------- ------ 9.8/11.6 MB 23.3 MB/s eta 0:00:01
   ---------------------------------------  11.6/11.6 MB 29.7 MB/s eta 0:00:01
   ---------------------------------------  11.6/11.6 MB 29.7 MB/s eta 0:00:01
   ---------------------------------------  11.6/11.6 MB 29.7 MB/s eta 0:00:01
   ---------------------------------------- 11.6/11.6 MB 22.6 MB/s eta 0:00:00
Using cached pytz-2024.2-py2.py3-none-any.whl (508 kB)
Using cached tzdata-2024.2-py2.py3-none-any.whl (346 kB)
Installing collected packages: pytz, tzdata, pandas, seaborn
Successfully installed pandas-2.2.3 pytz-2024.2 seaborn-0.13.2 tzdata-2024.2
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: C:\Users\egortega\AppData\Local\Microsoft\WindowsApps\P
ythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install
--upgrade pip
```

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import pandas as pd
```

[4]:
```
df = pd.read_csv("avocado.csv")
df
```

[4]:

| | Unnamed: 0 | Date | AveragePrice | Total Volume | 4046 | 4225 | \ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 2015-12-27 | 1.33 | 64236.62 | 1036.74 | 54454.85 | |
| 1 | 1 | 2015-12-20 | 1.35 | 54876.98 | 674.28 | 44638.81 | |
| 2 | 2 | 2015-12-13 | 0.93 | 118220.22 | 794.70 | 109149.67 | |
| 3 | 3 | 2015-12-06 | 1.08 | 78992.15 | 1132.00 | 71976.41 | |
| 4 | 4 | 2015-11-29 | 1.28 | 51039.60 | 941.48 | 43838.39 | |
| … | … | … | … | … | … | … | |
| 18244 | 7 | 2018-02-04 | 1.63 | 17074.83 | 2046.96 | 1529.20 | |
| 18245 | 8 | 2018-01-28 | 1.71 | 13888.04 | 1191.70 | 3431.50 | |
| 18246 | 9 | 2018-01-21 | 1.87 | 13766.76 | 1191.92 | 2452.79 | |
| 18247 | 10 | 2018-01-14 | 1.93 | 16205.22 | 1527.63 | 2981.04 | |
| 18248 | 11 | 2018-01-07 | 1.62 | 17489.58 | 2894.77 | 2356.13 | |

| | 4770 | Total Bags | Small Bags | Large Bags | XLarge Bags | type | \ |
|---|---|---|---|---|---|---|---|
| 0 | 48.16 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | |
| 1 | 58.33 | 9505.56 | 9408.07 | 97.49 | 0.0 | conventional | |
| 2 | 130.50 | 8145.35 | 8042.21 | 103.14 | 0.0 | conventional | |
| 3 | 72.58 | 5811.16 | 5677.40 | 133.76 | 0.0 | conventional | |
| 4 | 75.78 | 6183.95 | 5986.26 | 197.69 | 0.0 | conventional | |
| … | … | … | … | … | … | … | |

```
18244    0.00    13498.67    13066.82      431.85          0.0       organic
18245    0.00     9264.84     8940.04      324.80          0.0       organic
18246  727.94     9394.11     9351.80       42.31          0.0       organic
18247  727.01    10969.54    10919.54       50.00          0.0       organic
18248  224.53    12014.15    11988.14       26.01          0.0       organic

       year            region
0      2015            Albany
1      2015            Albany
2      2015            Albany
3      2015            Albany
4      2015            Albany
…      …               …
18244  2018  WestTexNewMexico
18245  2018  WestTexNewMexico
18246  2018  WestTexNewMexico
18247  2018  WestTexNewMexico
18248  2018  WestTexNewMexico

[18249 rows x 14 columns]
```

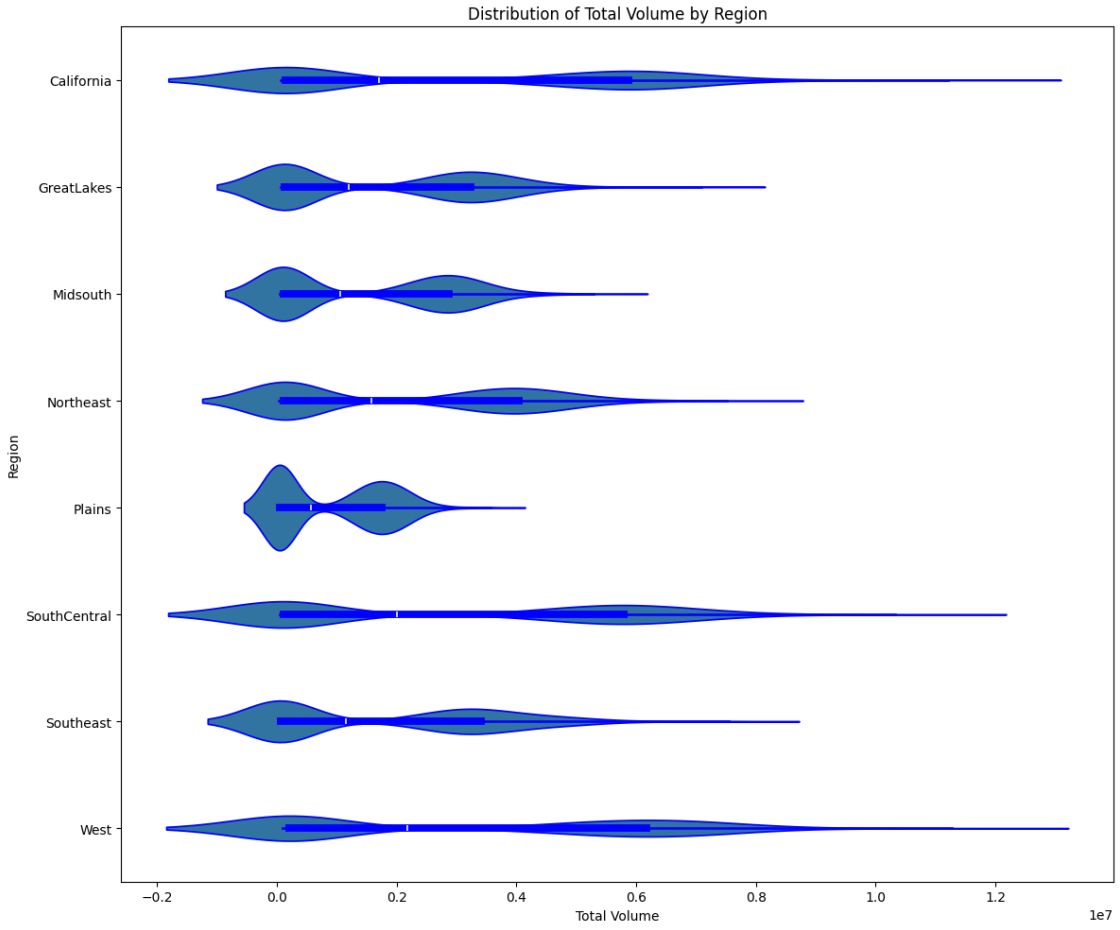
[5]:
```python
# Display Unique Region Names and their quantities
unique_regions = df['region'].value_counts()  # Get unique region names and their counts
print(unique_regions)  # Print unique region names and their counts
```

```
region
Albany                338
Atlanta               338
BaltimoreWashington   338
Boise                 338
Boston                338
BuffaloRochester      338
California            338
Charlotte             338
Chicago               338
CincinnatiDayton      338
Columbus              338
DallasFtWorth         338
Denver                338
Detroit               338
GrandRapids           338
GreatLakes            338
HarrisburgScranton    338
HartfordSpringfield   338
Houston               338
Indianapolis          338
Jacksonville          338
```

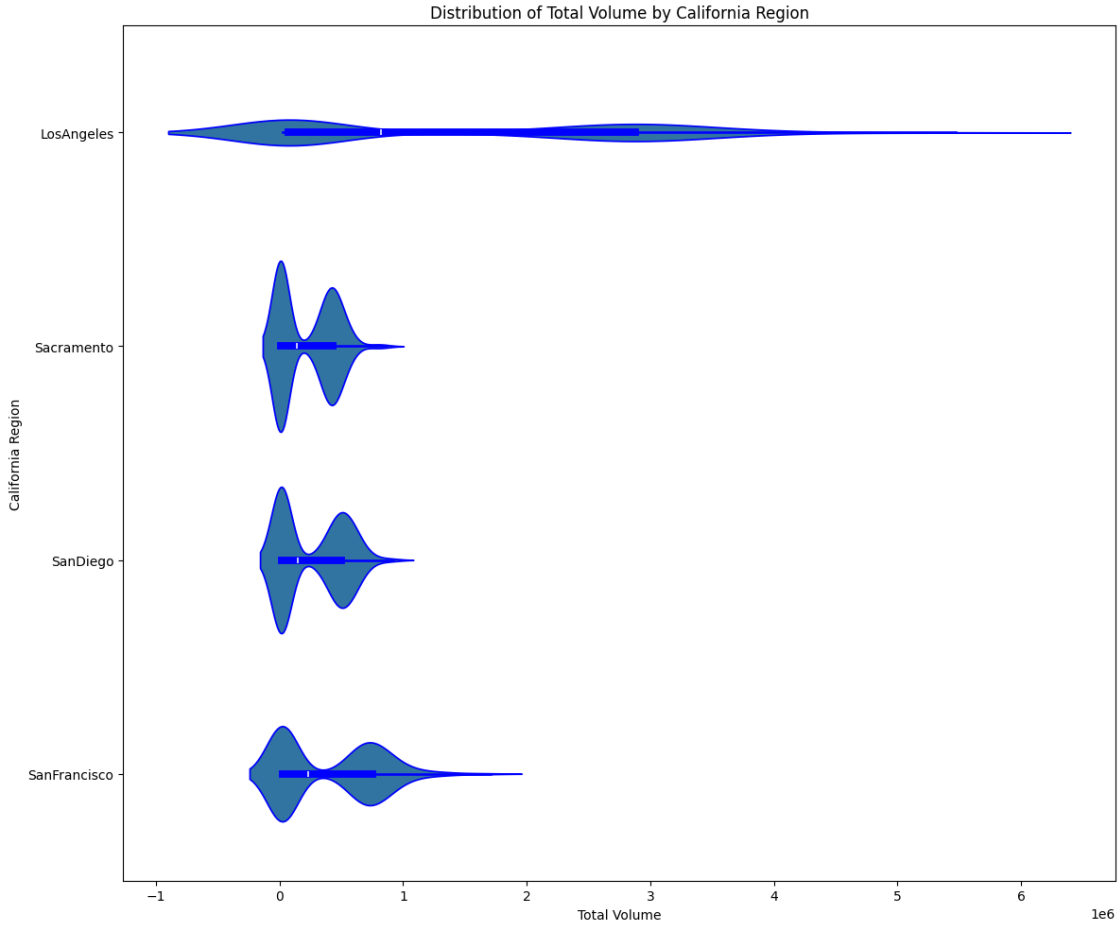
```
LasVegas               338
LosAngeles             338
Louisville             338
MiamiFtLauderdale      338
Midsouth               338
Nashville              338
NewOrleansMobile       338
NewYork                338
Northeast              338
NorthernNewEngland     338
Orlando                338
Philadelphia           338
PhoenixTucson          338
Pittsburgh             338
Plains                 338
Portland               338
RaleighGreensboro      338
RichmondNorfolk        338
Roanoke                338
Sacramento             338
SanDiego               338
SanFrancisco           338
Seattle                338
SouthCarolina          338
SouthCentral           338
Southeast              338
Spokane                338
StLouis                338
Syracuse               338
Tampa                  338
TotalUS                338
West                   338
WestTexNewMexico       335
Name: count, dtype: int64
```

[7]:
```
# Filter the DataFrame to include only the regions
regions = ['California', 'GreatLakes', 'Midsouth', 'Northeast', 'Plains',␣
 ↪'SouthCentral', 'Southeast', 'West']
filtered_df_regions = df[df['region'].isin(regions)]

plt.figure(figsize=(12,10))
sns.violinplot(x="Total Volume", y="region", data=filtered_df_regions,␣
 ↪edgecolor="blue")
plt.title("Distribution of Total Volume by Region")
plt.xlabel("Total Volume")
plt.ylabel("Region")
```

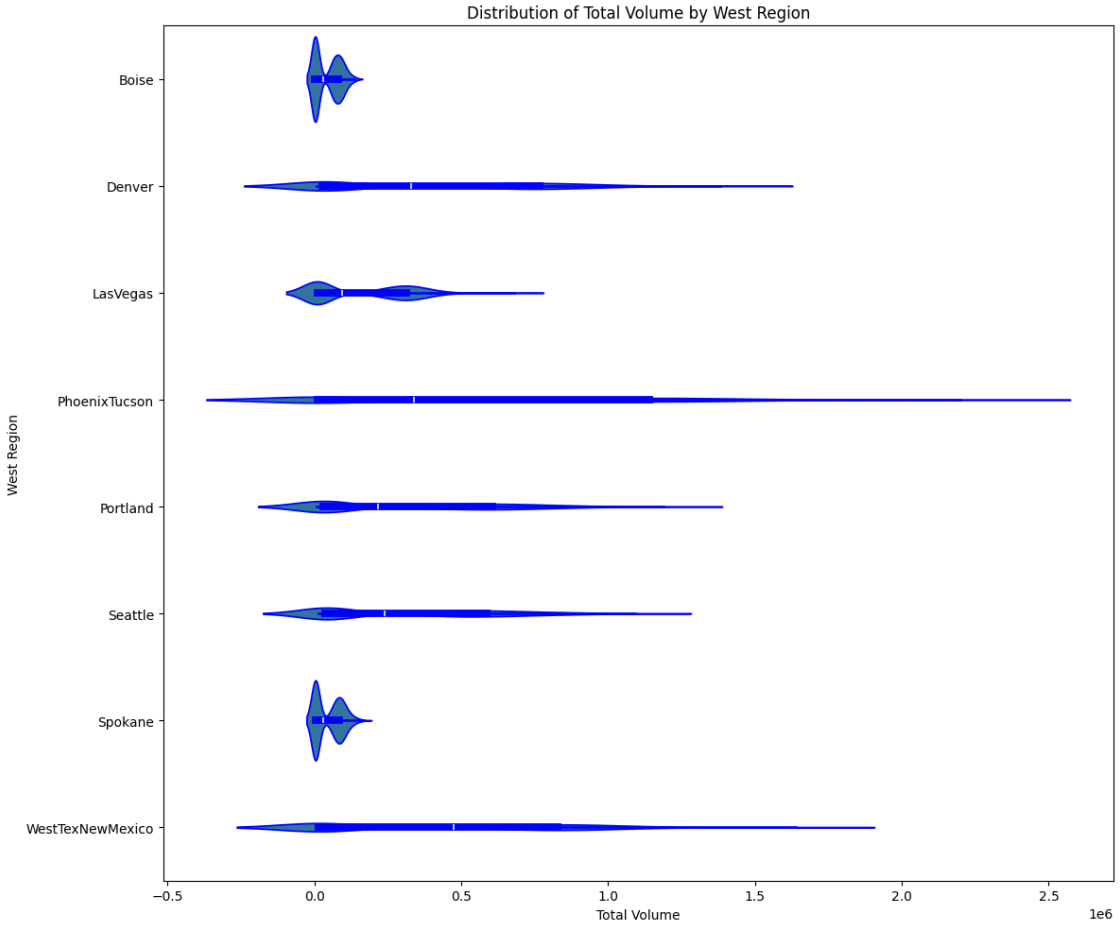
```
plt.tight_layout()
plt.show()
```



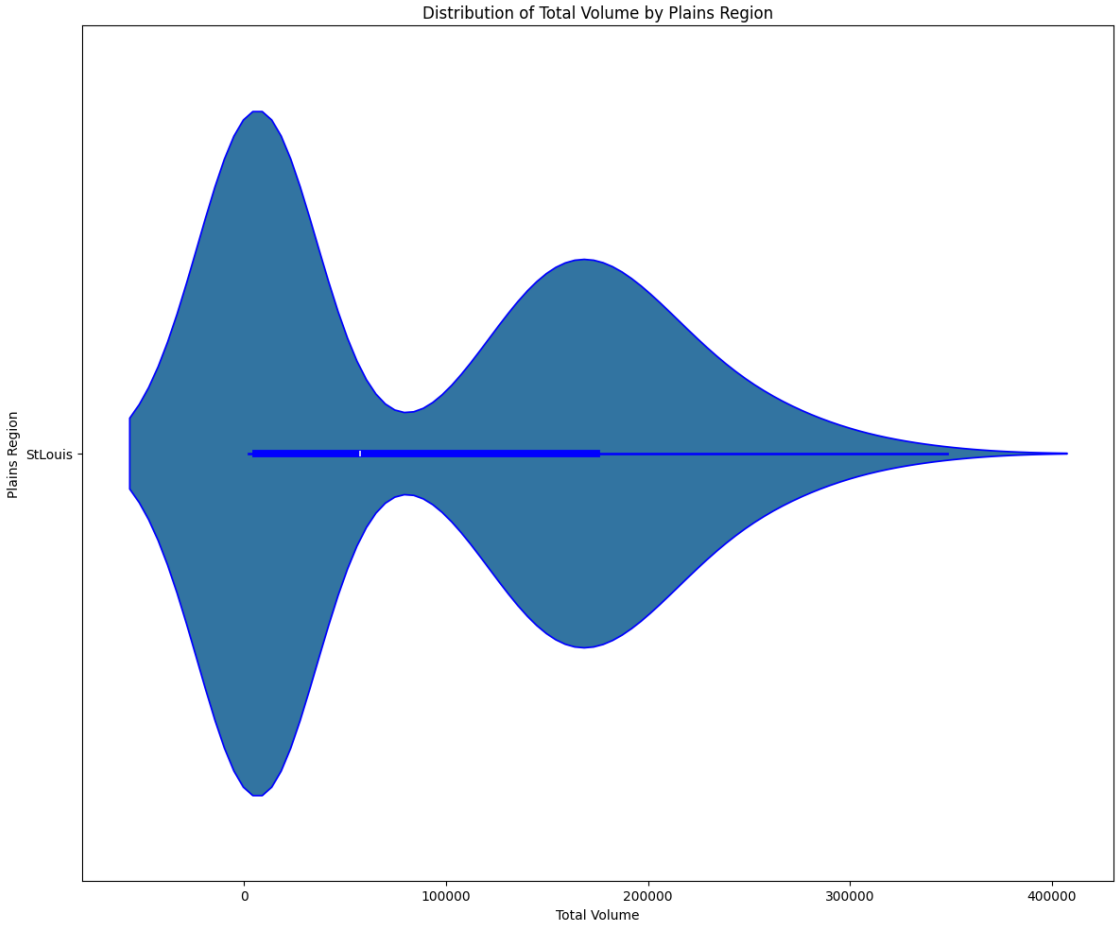

[10]:
```
# Filter the DataFrame to include only the California region
california_region = ['LosAngeles', 'Sacramento', 'SanDiego', 'SanFrancisco']
filtered_df_regions_california = df[df['region'].isin(california_region)]

plt.figure(figsize=(12,10))
sns.violinplot(x="Total Volume", y="region",␣
 ↪data=filtered_df_regions_california, edgecolor="blue")
plt.title("Distribution of Total Volume by California Region")
plt.xlabel("Total Volume")
plt.ylabel("California Region")

plt.tight_layout()
plt.show()
```




[11]:
```
# Filter the DataFrame to include only the California region
west_region = ['Boise', 'Denver', 'LasVegas', 'PhoenixTucson', 'Portland',␣
 ↪'Seattle', 'Spokane', 'WestTexNewMexico']
filtered_df_regions_west = df[df['region'].isin(west_region)]

plt.figure(figsize=(12,10))
sns.violinplot(x="Total Volume", y="region", data=filtered_df_regions_west,␣
 ↪edgecolor="blue")
plt.title("Distribution of Total Volume by West Region")
plt.xlabel("Total Volume")
plt.ylabel("West Region")

plt.tight_layout()
plt.show()
```

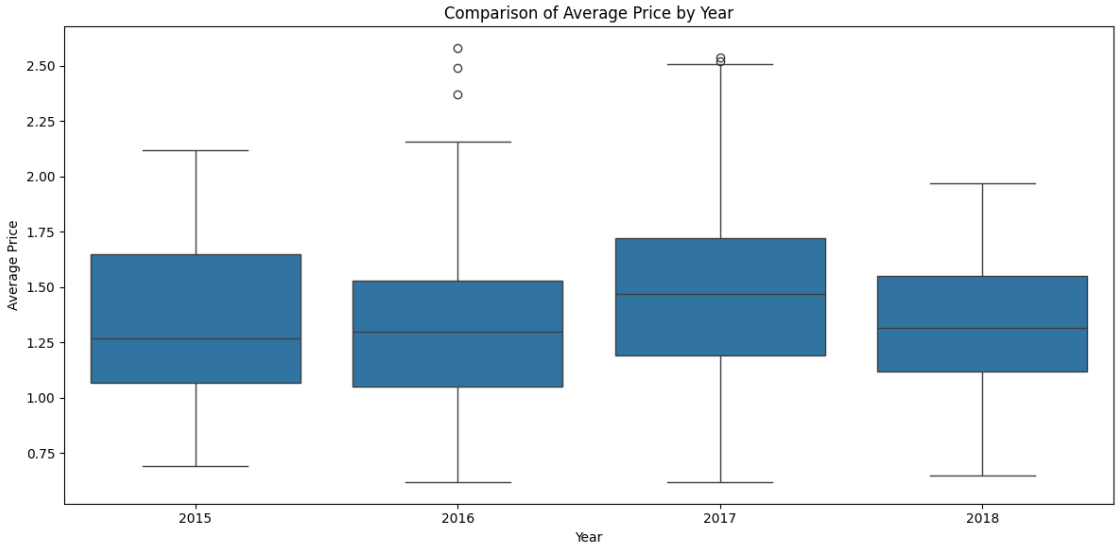



[12]:
```
# Filter the DataFrame to include only the California region
plains_region = ['StLouis']
filtered_df_regions_plains = df[df['region'].isin(plains_region)]

plt.figure(figsize=(12,10))
sns.violinplot(x="Total Volume", y="region", data=filtered_df_regions_plains,␣
 ↪edgecolor="blue")
plt.title("Distribution of Total Volume by Plains Region")
plt.xlabel("Total Volume")
plt.ylabel("Plains Region")

plt.tight_layout()
plt.show()
```

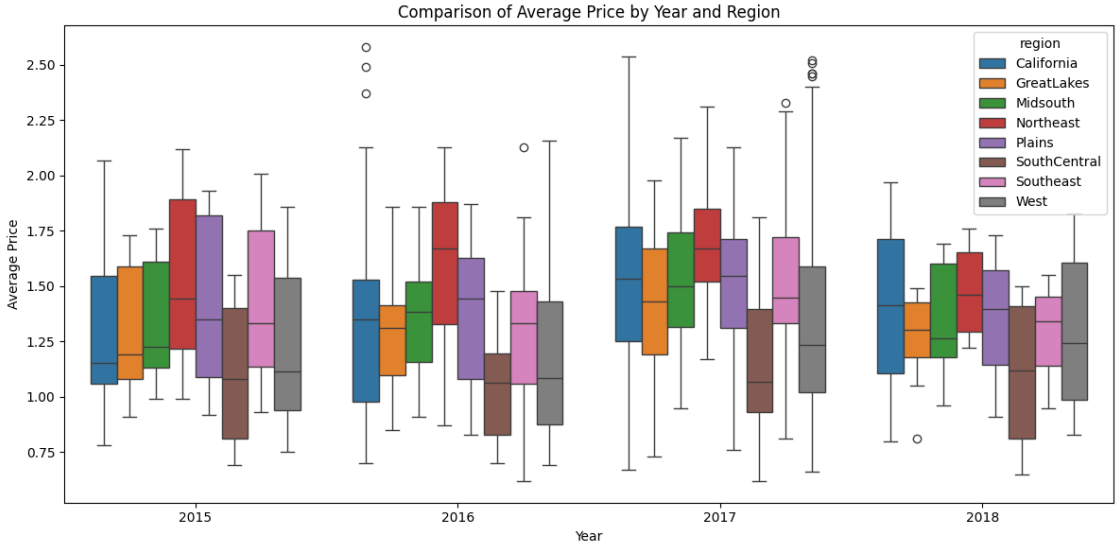



2. **Boxplot Comparativo de Precios entre Años:**
   - **Uso de Datos:** Usa las columnas `AveragePrice` y `year`.
   - **Esperado:** Genera boxplots para comparar la distribución de precios.
     - Utiliza `boxplot` de `seaborn` para crear boxplots que comparen `AveragePrice` entre diferentes años.
     - Asegúrate de que cada boxplot represente un año diferente.
     - Incluye etiquetas y títulos descriptivos usando `plt.title()`.

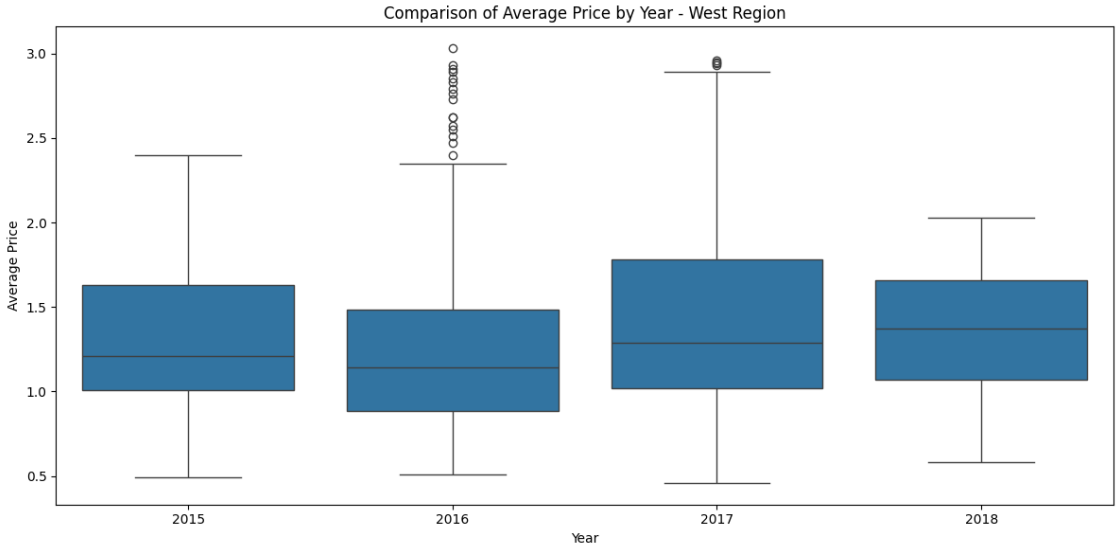
[13]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x="year", y="AveragePrice", data=filtered_df_regions)
plt.title("Comparison of Average Price by Year")
plt.xlabel("Year")
plt.ylabel("Average Price")

plt.tight_layout()
plt.show()
```



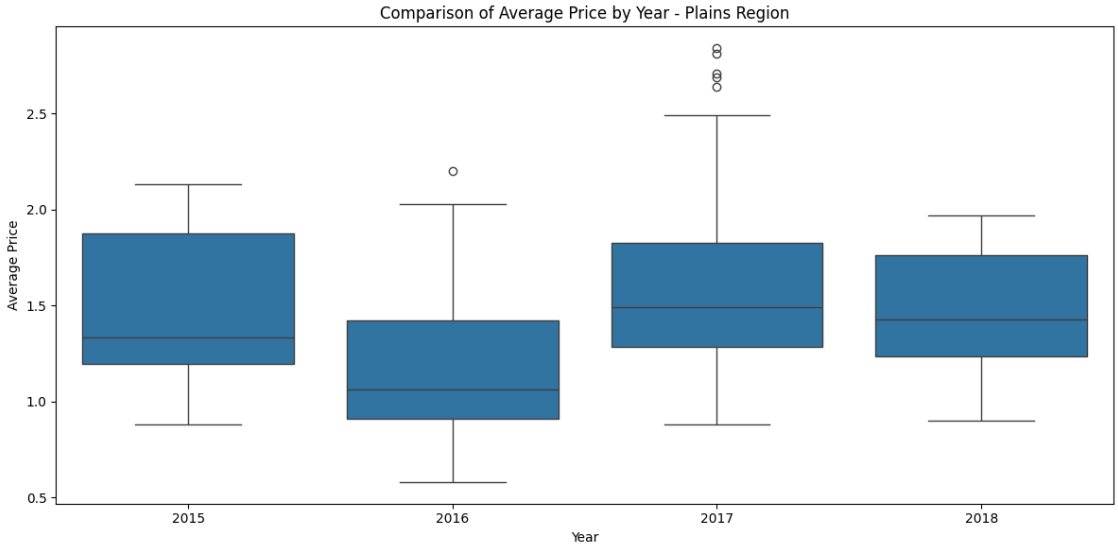

[14]:
```
# Plot the comparison of average price by year and region
plt.figure(figsize=(12, 6))
sns.boxplot(x="year", y="AveragePrice", hue="region", data=filtered_df_regions)
plt.title("Comparison of Average Price by Year and Region")
plt.xlabel("Year")
plt.ylabel("Average Price")

plt.tight_layout()
plt.show()
```



[15]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x="year", y="AveragePrice", data=filtered_df_regions_west)
plt.title("Comparison of Average Price by Year - West Region")
plt.xlabel("Year")
plt.ylabel("Average Price")

plt.tight_layout()
plt.show()
```

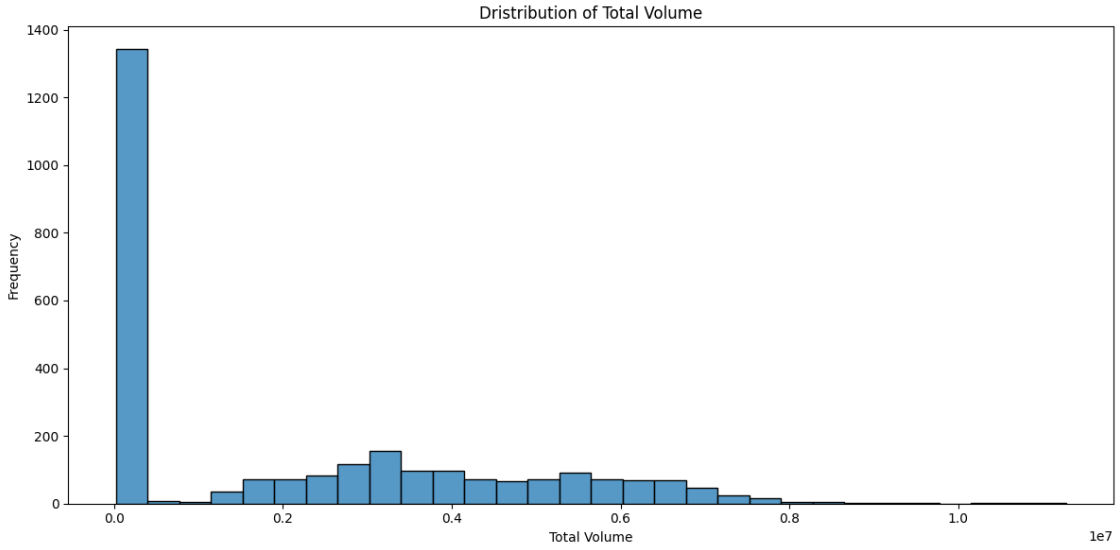



[16]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x="year", y="AveragePrice", data=filtered_df_regions_plains)
plt.title("Comparison of Average Price by Year - Plains Region")
plt.xlabel("Year")
plt.ylabel("Average Price")

plt.tight_layout()
plt.show()
```

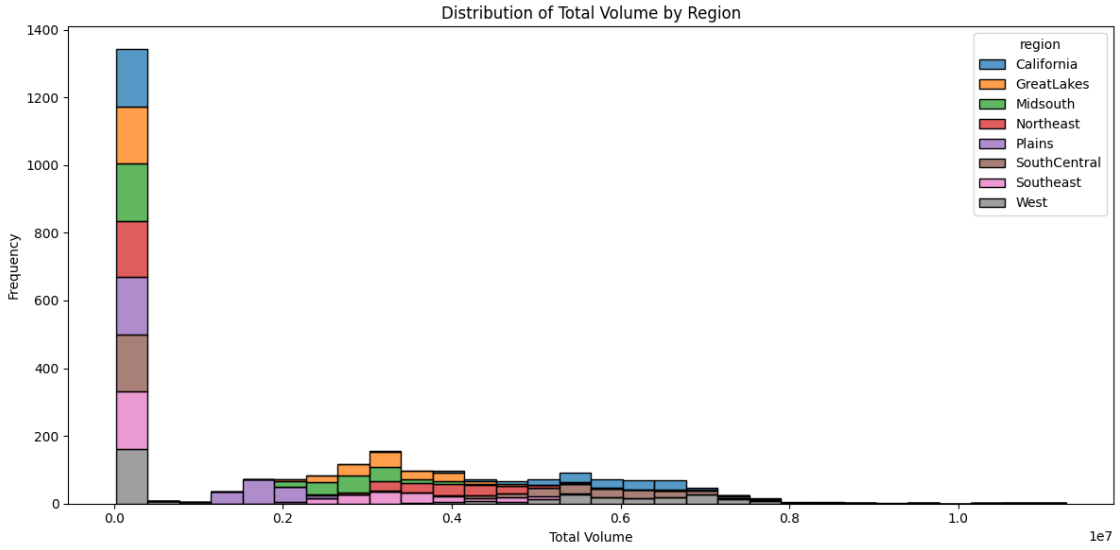



3. **Histograma de Volumen Total de Ventas:**
   - **Uso de Datos:** Usa la columna `Total Volume`.
   - **Esperado:** Crea un histograma para mostrar la distribución del volumen total de ventas.
     - Utiliza `hist()` de `matplotlib` para crear el histograma.
     - Ajusta el número de bins para una visualización clara usando el parámetro `bins`.
     - Añade etiquetas y un título que describa lo que se muestra.

[17]:
```
plt.figure(figsize=(12, 6))
sns.histplot(filtered_df_regions["Total Volume"], bins=30, edgecolor="black")
plt.title("Dristribution of Total Volume")
plt.xlabel("Total Volume")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

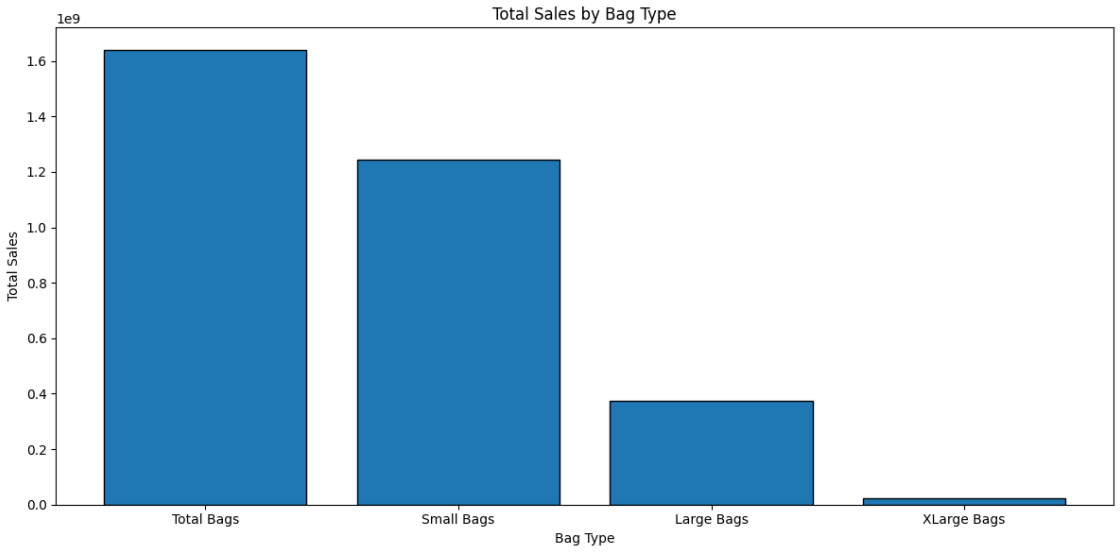



[18]:
```
plt.figure(figsize=(12, 6))
sns.histplot(data=filtered_df_regions, x="Total Volume", bins=30,␣
 ↪edgecolor="black", hue="region", multiple="stack")
plt.title("Distribution of Total Volume by Region")
plt.xlabel("Total Volume")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```



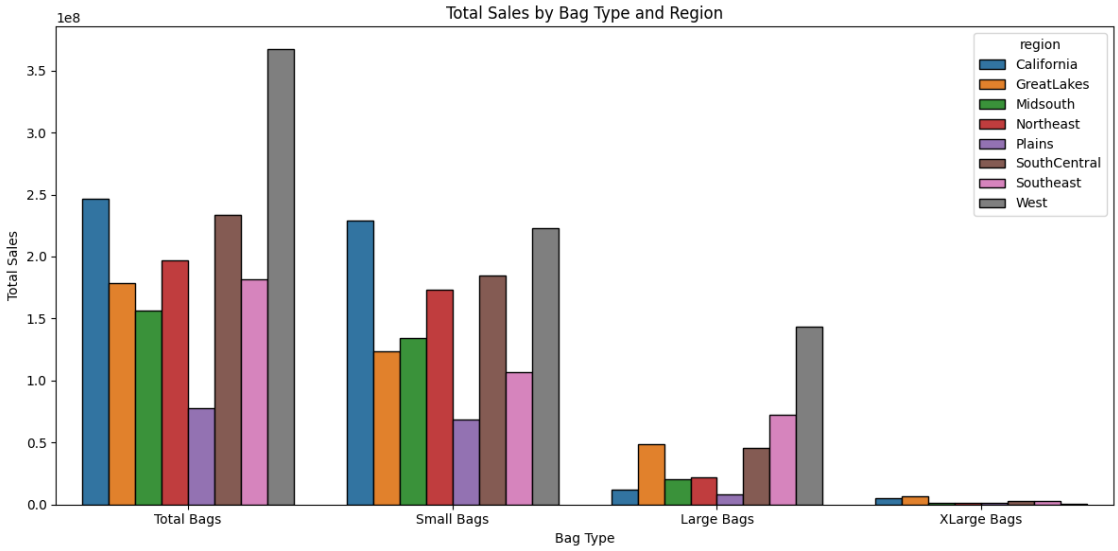
4. **Gráfico de Barras de Ventas por Tipo de Bolsa:**
   - **Uso de Datos:** Utiliza las columnas `Total Bags`, `Small Bags`, `Large Bags` y `XLarge Bags`.
   - **Esperado:** Compara las ventas de diferentes tipos de bolsas.
     - Suma los volúmenes de ventas por tipo de bolsa utilizando `sum()`.
     - Crea un gráfico de barras con `plt.bar()` para mostrar las diferencias en ventas.
     - Asegúrate de incluir etiquetas para cada tipo de bolsa.

[19]:
```
total_bags = filtered_df_regions["Total Bags"].sum()
small_bags = filtered_df_regions["Small Bags"].sum()
large_bags = filtered_df_regions["Large Bags"].sum()
xlarge_bags = filtered_df_regions["XLarge Bags"].sum()

bag_types = ["Total Bags", "Small Bags", "Large Bags", "XLarge Bags"]
sales = [total_bags, small_bags, large_bags, xlarge_bags]

plt.figure(figsize=(12, 6))
plt.bar(bag_types, sales, edgecolor='black')

plt.title('Total Sales by Bag Type')
plt.xlabel('Bag Type')
plt.ylabel('Total Sales')

plt.tight_layout()
plt.show()
```




[20]:
```
# Calculate the total sales by bag type for each region
total_bags_region = filtered_df_regions.groupby('region')['Total Bags'].sum().
↪reset_index()
```

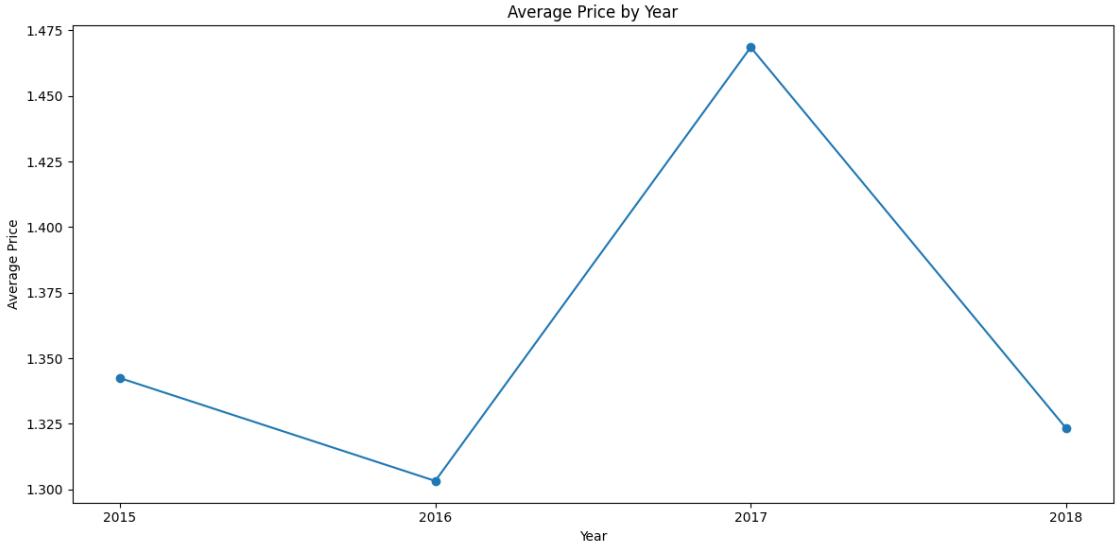
```
small_bags_region = filtered_df_regions.groupby('region')['Small Bags'].sum().
 ↪reset_index()
large_bags_region = filtered_df_regions.groupby('region')['Large Bags'].sum().
 ↪reset_index()
xlarge_bags_region = filtered_df_regions.groupby('region')['XLarge Bags'].sum().
 ↪reset_index()

# Merge the data into a single DataFrame
bag_sales = total_bags_region.merge(small_bags_region, on='region').
 ↪merge(large_bags_region, on='region').merge(xlarge_bags_region, on='region')
bag_sales = bag_sales.melt(id_vars='region', var_name='Bag Type',␣
 ↪value_name='Total Sales')

# Plot the total sales by bag type divided by regions
plt.figure(figsize=(12, 6))
sns.barplot(x='Bag Type', y='Total Sales', hue='region', data=bag_sales,␣
 ↪edgecolor='black')

plt.title('Total Sales by Bag Type and Region')
plt.xlabel('Bag Type')
plt.ylabel('Total Sales')

plt.tight_layout()
plt.show()
```




5. **Gráfico de Líneas de Precios Promedios por Año:**
   - **Uso de Datos:** Utiliza las columnas `AveragePrice` y `year`.
   - **Esperado:** Visualiza la tendencia de precios promedio a lo largo de los años.

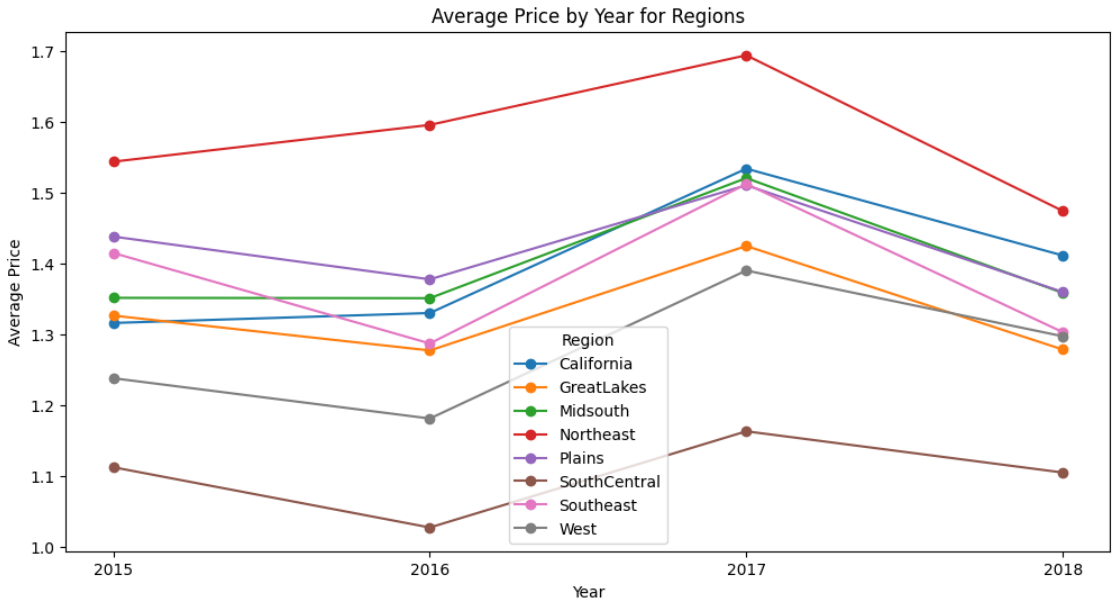
- Agrupa los datos por `year` y calcula el promedio de `AveragePrice`.
- Usa `plt.plot()` para crear un gráfico de líneas que muestre la evolución de precios.
- Añade un título y etiquetas descriptivas a los ejes usando `plt.title()` y `plt.xlabel()`.

[21]:
```
average_price_per_year = filtered_df_regions.groupby("year")["AveragePrice"].
 ↪mean()

plt.figure(figsize=(12, 6))
# Plotting the average price per year with markers and a line
plt.plot(average_price_per_year.index, average_price_per_year.values,␣
 ↪marker='o', linestyle='-')
plt.title("Average Price by Year")
plt.xlabel("Year")
plt.xticks(ticks=average_price_per_year.index, labels=average_price_per_year.
 ↪index.astype(int))
plt.ylabel("Average Price")

plt.tight_layout()
plt.show()
```




[22]:
```
# Group by year and region, then calculate the mean of the average price
average_price_per_year_region = filtered_df_regions.groupby(['year',␣
 ↪'region'])['AveragePrice'].mean().unstack()

# Plotting the average price per year for each region
plt.figure(figsize=(12, 6))
```

```
# Plot each region's average price per year
for region in regions:
    plt.plot(average_price_per_year_region.index,␣
 ↪average_price_per_year_region[region], marker='o', linestyle='-',␣
 ↪label=region)

plt.title("Average Price by Year for Regions")
plt.xlabel("Year")
plt.xticks(ticks=average_price_per_year_region.index,␣
 ↪labels=average_price_per_year_region.index.astype(int))
plt.ylabel("Average Price")
plt.legend(title='Region')
plt.show()
```



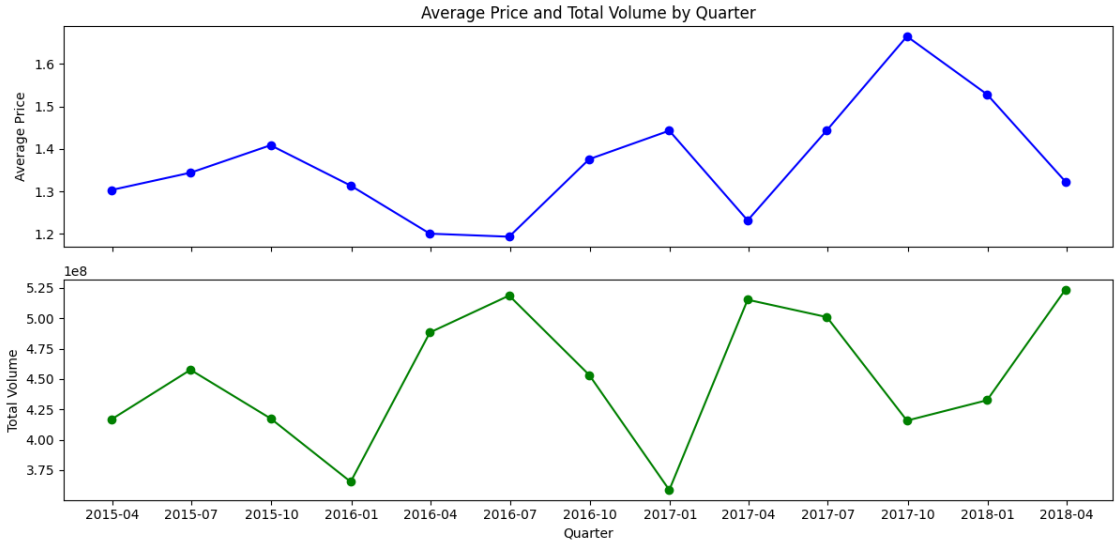

### 1.3.7 4. Análisis de Cohortes

**Resumen:** El análisis de cohortes permite agrupar datos según características específicas y observar cómo se comportan a lo largo del tiempo. Se centra en cohortes de precios y ventas para entender las dinámicas del mercado.

1. **Cohortes Basadas en Precios Promedios Trimestrales:**
   - **Uso de Datos:** Usa las columnas `AveragePrice`, `Total Volume` y `Date`.
   - **Esperado:** Crea cohortes trimestrales y analiza cambios en precios y volúmenes.
     - Agrupa los datos por trimestre usando `pd.Grouper` con `freq='Q'`.
     - Calcula el promedio de `AveragePrice` y suma `Total Volume` para cada cohorte.
     - Visualiza los resultados en un gráfico de líneas que muestre la evolución de las cohortes.

[23]:
```
filtered_df_regions["Date"] = pd.to_datetime(filtered_df_regions["Date"])

# Group by quarter and calculate the average of "AveragePrice" and the sum of␣
 ↪"Total Volume"
quarterly_data = filtered_df_regions.groupby(pd.Grouper(key="Date", freq="QE")).
 ↪agg({"AveragePrice": "mean", "Total Volume": "sum"}).reset_index()

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6))

ax1.plot(quarterly_data["Date"], quarterly_data["AveragePrice"], marker="o",␣
 ↪linestyle="-", color="b")
ax1.set_title("Average Price and Total Volume by Quarter")
ax1.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
ax1.set_xticklabels([])  # Hide x-axis labels for the first subplot
#ax1.set_xlabel("Quarter")
ax1.set_ylabel("Average Price")

ax2.plot(quarterly_data["Date"], quarterly_data["Total Volume"], marker="o",␣
 ↪linestyle="-", color="g")
#ax2.set_title("Total Volume by Quarter")
ax2.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
ax2.set_xlabel("Quarter")
ax2.set_ylabel("Total Volume")

plt.tight_layout()
plt.show()
```

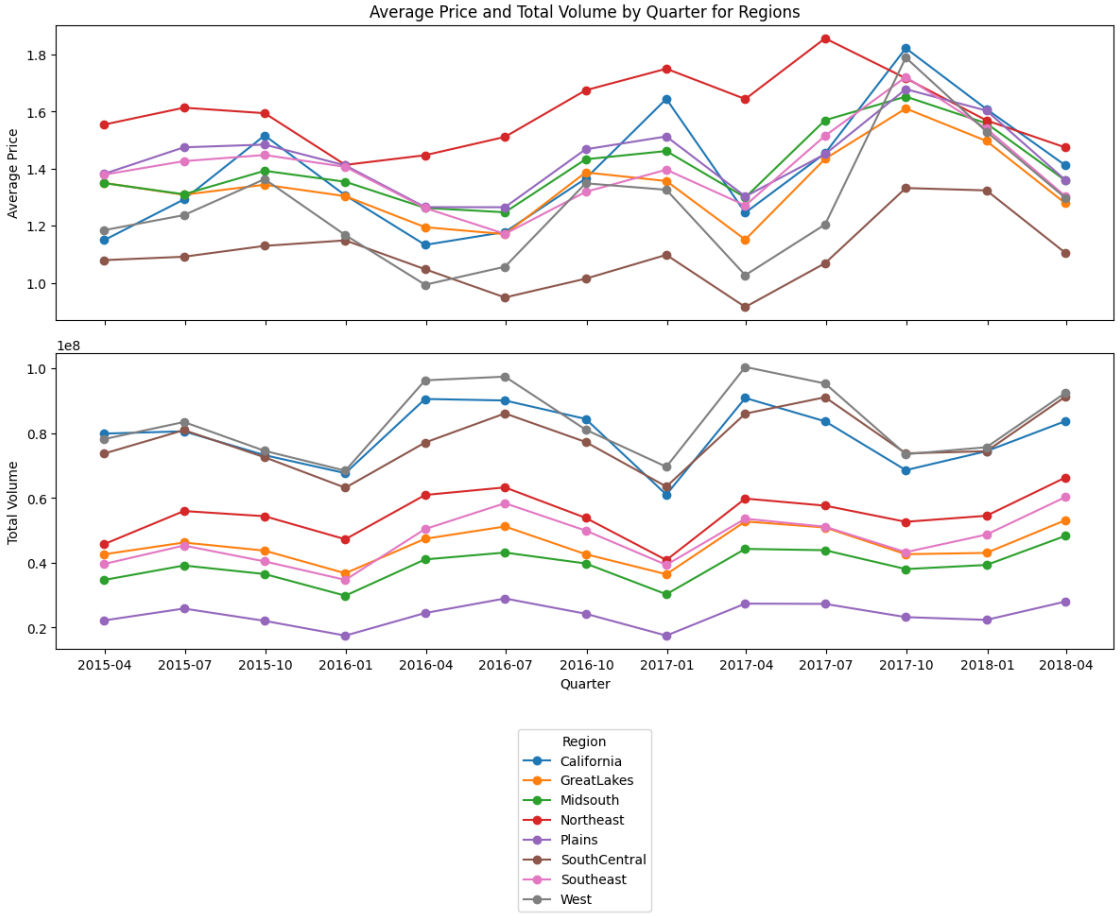
```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\810970783.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions["Date"] = pd.to_datetime(filtered_df_regions["Date"])
```




[24]:
```
# Convert the Date column to datetime
filtered_df_regions["Date"] = pd.to_datetime(filtered_df_regions["Date"])

# Group by quarter and region, then calculate the average of "AveragePrice" and␣
 ↪the sum of "Total Volume"
quarterly_data = filtered_df_regions.groupby([pd.Grouper(key="Date",␣
 ↪freq="QE"), "region"]).agg({"AveragePrice": "mean", "Total Volume": "sum"}).
 ↪reset_index()

# Plotting the average price and total volume by quarter for each region
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

# Plot each region's average price per quarter
for region in regions:
    region_data = quarterly_data[quarterly_data['region'] == region]
    ax1.plot(region_data["Date"], region_data["AveragePrice"], marker="o",␣
 ↪linestyle="-", label=region)

ax1.set_title("Average Price and Total Volume by Quarter for Regions")
ax1.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
ax1.set_xticklabels([])  # Hide x-axis labels for the first subplot
ax1.set_ylabel("Average Price")
#ax1.legend(title='Region')

# Plot each region's total volume per quarter
for region in regions:
    region_data = quarterly_data[quarterly_data['region'] == region]
```

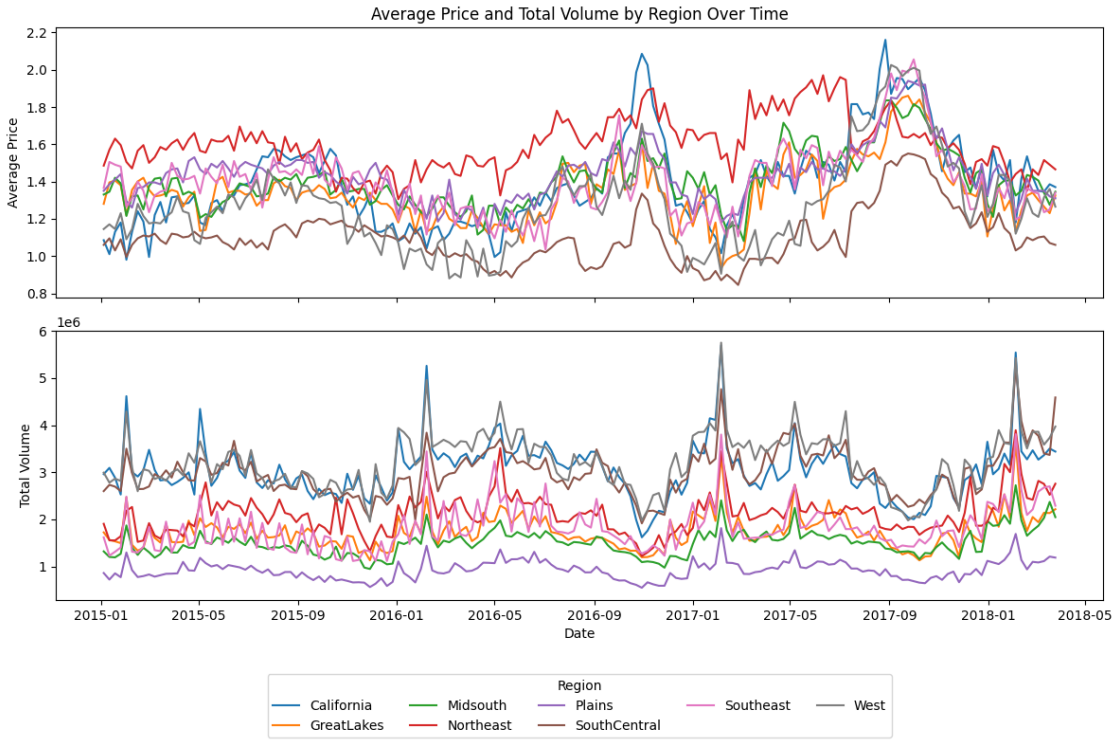
```
    ax2.plot(region_data["Date"], region_data["Total Volume"], marker="o",␣
 ↪linestyle="-", label=region)

#ax2.set_title("Total Volume by Quarter for Selected Regions")
ax2.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
ax2.set_xlabel("Quarter")
ax2.set_ylabel("Total Volume")
ax2.legend(title="Region", loc="upper center", bbox_to_anchor=(0.5, -0.25))

plt.tight_layout()
plt.show()
```

```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\3453302182.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions["Date"] = pd.to_datetime(filtered_df_regions["Date"])
```



2. **Cohortes por Región y Fecha:**
   - **Uso de Datos:** Utiliza las columnas `AveragePrice`, `Total Volume`, `region` y `Date`.
   - **Esperado:** Analiza cómo varían las cohortes de diferentes regiones.
     - Agrupa los datos por `region` y `Date` usando `groupby()`.
     - Calcula el promedio de precios y volumen para cada cohorte.
     - Presenta los resultados en gráficos de barras que muestren comparaciones entre regiones.

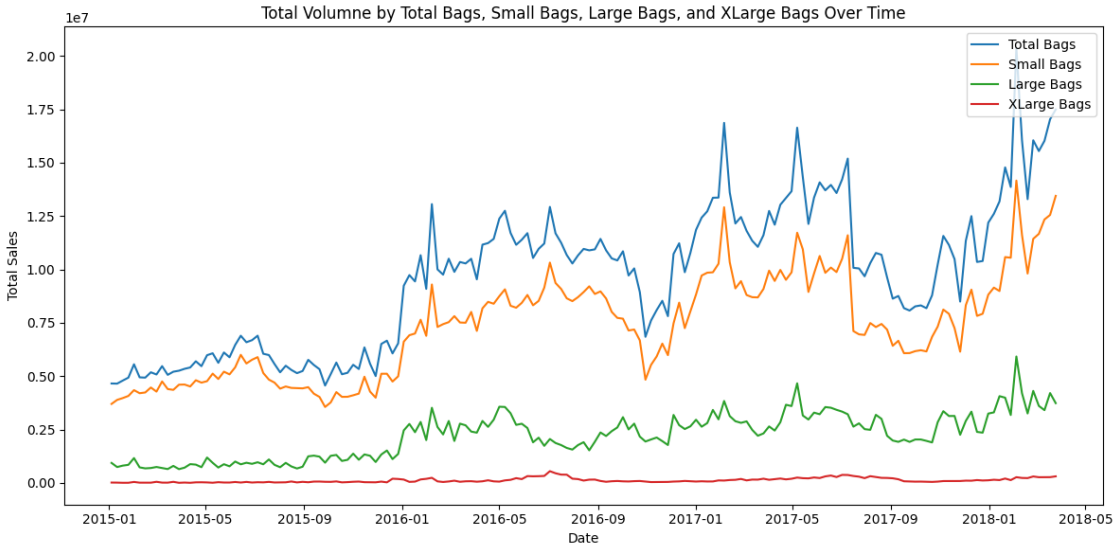
[25]:
```
region_date_grouped = filtered_df_regions.groupby(["region", "Date"]).agg({
    "AveragePrice": "mean",
    "Total Volume": "mean"
}).reset_index()

fig2, (plt1, plt2) = plt.subplots(2, 1, figsize=(12, 8))

# Plotting Average Price comparison by region over time
#plt1.figure(figsize=(14, 8))
for region in region_date_grouped["region"].unique():
    region_data = region_date_grouped[region_date_grouped["region"] == region]
    plt1.plot(region_data["Date"], region_data["AveragePrice"], label=region)

#plt.xlabel("Date")
plt1.set_title("Average Price and Total Volume by Region Over Time")
plt1.set_xticklabels([])  # Hide x-axis labels for the first subplot
plt1.set_ylabel("Average Price")
#plt.legend(title="Region")
#plt.xticks(rotation=45)
#plt1.tight_layout()
#plt1.show()

# Plotting Total Volume comparison by region over time
#plt2.figure(figsize=(14, 8))
for region in region_date_grouped["region"].unique():
    region_data = region_date_grouped[region_date_grouped["region"] == region]
    plt2.plot(region_data["Date"], region_data["Total Volume"], label=region)

plt2.set_xlabel("Date")
plt2.set_ylabel("Total Volume")
#plt.title("Total Volume by Region Over Time")
plt2.legend(title="Region", loc="upper center", bbox_to_anchor=(0.5, -0.25),␣
 ↪ncol=5)
#plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```




3. **Análisis de Cohortes en Función del Tipo de Bolsa:**
   - **Uso de Datos:** Usa las columnas `Total Bags`, `Small Bags`, `Large Bags`, `XLarge Bags` y `Date`.
   - **Esperado:** Examina cómo se comportan las diferentes cohortes según el tipo de bolsa.
     - Agrupa los datos por tipo de bolsa y `Date`.
     - Calcula el volumen de ventas total y muestra los resultados en un gráfico de líneas.

[ ]:
```
# Group by quarter and calculate the total sales volume for each type of bag.
bags_by_date = filtered_df_regions.groupby("Date").agg({
    "Total Bags": "sum",
    "Small Bags": "sum",
    "Large Bags": "sum",
    "XLarge Bags": "sum"
}).reset_index()

plt.figure(figsize=(12, 6))
plt.plot(bags_by_date["Date"], bags_by_date["Total Bags"], linestyle='-',␣
 ↪label="Total Bags")
plt.plot(bags_by_date["Date"], bags_by_date["Small Bags"], linestyle='-',␣
 ↪label="Small Bags")
plt.plot(bags_by_date["Date"], bags_by_date["Large Bags"], linestyle='-',␣
 ↪label="Large Bags")
```

```
plt.plot(bags_by_date["Date"], bags_by_date["XLarge Bags"], linestyle='-',␣
 ↪label="XLarge Bags")

plt.title("Total Volumne by Total Bags, Small Bags, Large Bags, and XLarge Bags␣
 ↪Over Time")
plt.xlabel("Date")
plt.ylabel("Total Sales")
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()
```

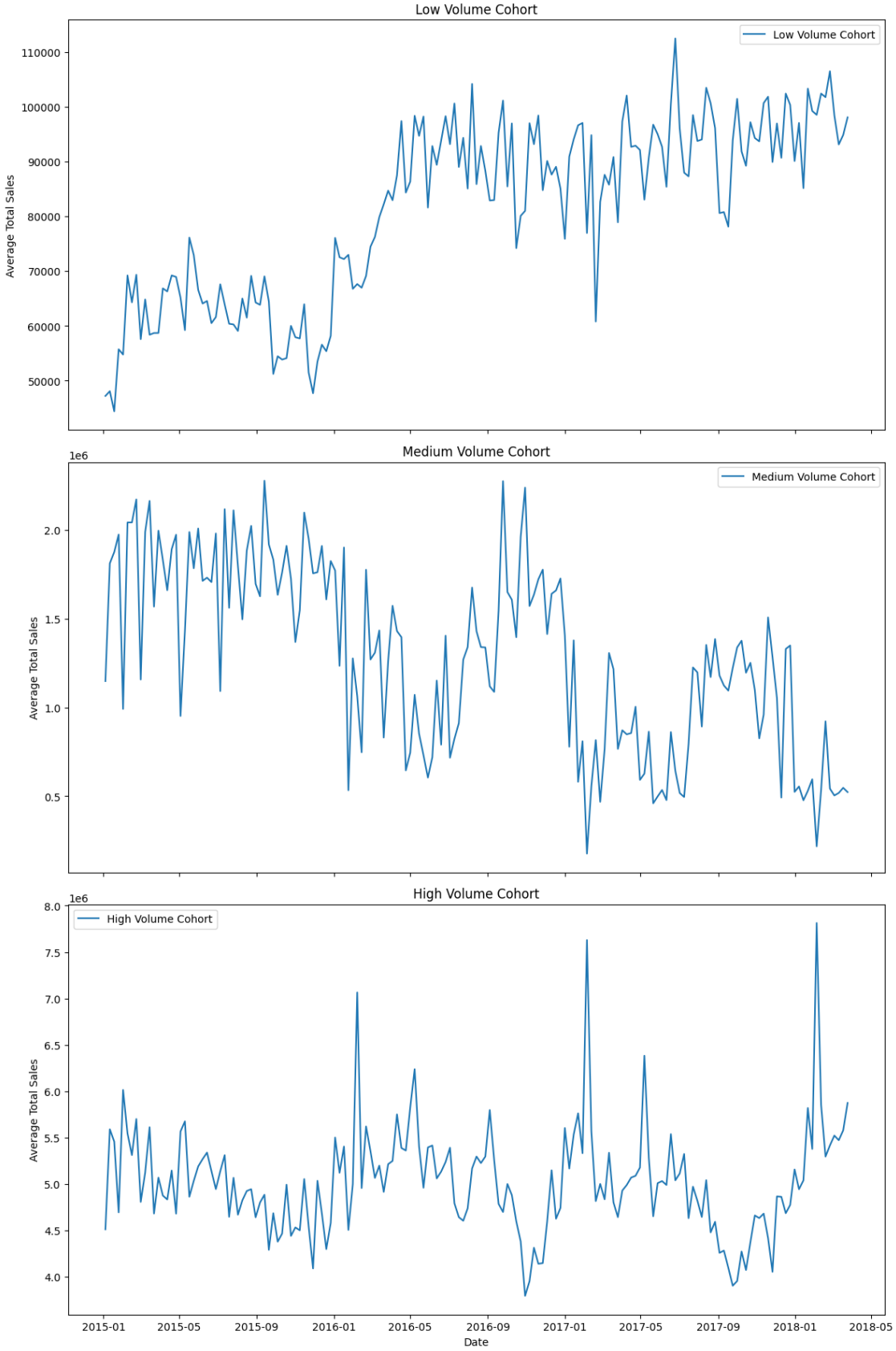



4. **Cohortes de Clientes Basadas en Ventas:**
   - **Uso de Datos:** Usa las columnas `Total Volume`, `Date` y `region`.
   - **Esperado:** Analiza el comportamiento de las cohortes según el volumen de ventas.
     - Clasifica los clientes según su volumen de compras.
     - Visualiza las cohortes en gráficos de líneas o barras que muestren el comportamiento de compra a lo largo del tiempo.

[27]:
```
# Define cohorts based on Total Volume using quantiles
filtered_df_regions["Volume Cohort"] = pd.qcut(filtered_df_regions["Total␣
 ↪Volume"], q=[0, 0.33, 0.66, 1], labels=["Low", "Medium", "High"])

# Group data by "Volume Cohort" and "Date" to get the average sales volume for␣
 ↪each cohort over time
cohort_sales = filtered_df_regions.groupby(["Volume Cohort", "Date"]).agg({
    "Total Volume": "mean"
}).reset_index()
```

```
#plt.figure(figsize=(12, 8))
# Create a figure with 3 subplots arranged vertically
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 18), sharex=True)

# Plot each volume cohort over time to observe purchase behavior
for i, cohort in enumerate(cohort_sales["Volume Cohort"].unique()):
    cohort_data = cohort_sales[cohort_sales["Volume Cohort"] == cohort]
    axes[i].plot(cohort_data["Date"], cohort_data["Total Volume"],␣
 ↪label=f"{cohort} Volume Cohort")
    axes[i].set_ylabel("Average Total Sales")
    axes[i].set_title(f"{cohort} Volume Cohort")
    axes[i].legend()

# Set common labels
axes[-1].set_xlabel("Date")

# Plot each volume cohort over time to observe purchase behavior
#for cohort in cohort_sales["Volume Cohort"].unique():
#    cohort_data = cohort_sales[cohort_sales["Volume Cohort"] == cohort]
#    plt.plot(cohort_data["Date"], cohort_data["Total Volume"],␣
 ↪label=f"{cohort} Volume Cohort")

#plt.xlabel("Date")
#plt.ylabel("Average Total Volume")
#plt.title("Customer Cohorts Based on Sales Volume Over Time")
#plt.legend(title="Volume Cohort")
#plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\1588898550.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions["Volume Cohort"] = pd.qcut(filtered_df_regions["Total
Volume"], q=[0, 0.33, 0.66, 1], labels=["Low", "Medium", "High"])
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\1588898550.py:5:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  cohort_sales = filtered_df_regions.groupby(["Volume Cohort", "Date"]).agg({
```

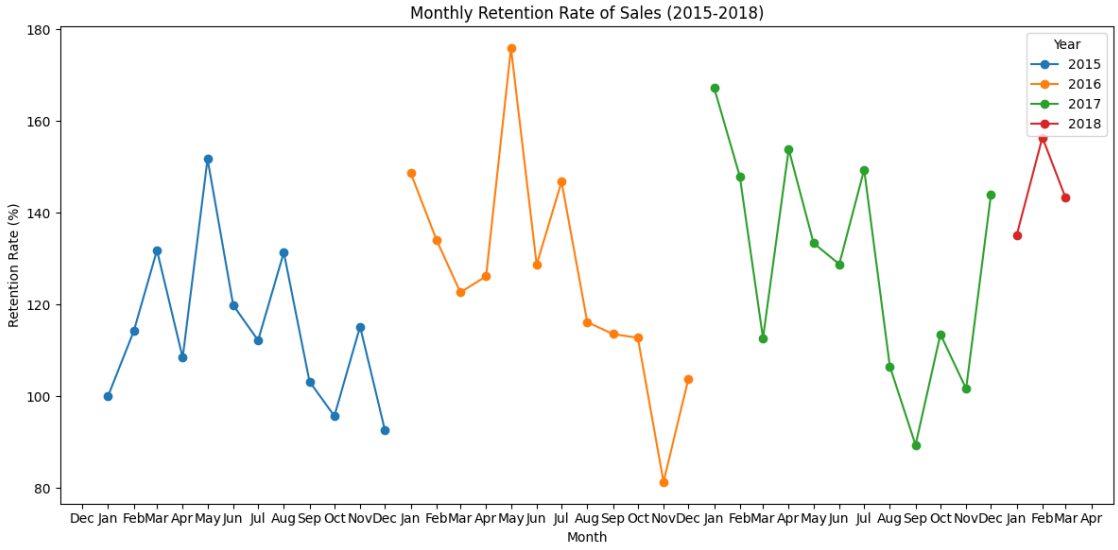

Low Volume Cohort
Low Volume Cohort
Average Total Sales
110000
100000
90000
80000
70000
60000
50000
Medium Volume Cohort
1e6
Medium Volume Cohort
Average Total Sales
2.0
1.5
1.0
0.5
High Volume Cohort
1e6
High Volume Cohort
Average Total Sales
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05
Date

5. **Evaluación de Retención de Ventas por Cohorte:**
   - **Uso de Datos:** Usa las columnas `Total Volume` y `Date`.
   - **Esperado:** Estudia cómo se retienen las ventas en cohortes a lo largo de un año.
     - Agrupa los datos por mes y cohortes.
     - Calcula la retención de ventas y visualiza los resultados en un gráfico de líneas que muestre las tasas de retención.

[48]:
```
# Create a column for month and year
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')

# Group by month and calculate the total sales volume for the entire dataset
monthly_sales = filtered_df_regions.groupby("Month")["Total Volume"].sum().
 ↪reset_index()

# Define sales cohorts in the first month of the entire dataset as a baseline
first_month_sales = monthly_sales["Total Volume"].iloc[0]

# Calculate sales retention rate: sales volume each month as a percentage of␣
 ↪the first month
monthly_sales["Retention Rate"] = (monthly_sales["Total Volume"] /␣
 ↪first_month_sales) * 100

# Convert Month to a date for the chart
monthly_sales["Month"] = monthly_sales["Month"].dt.to_timestamp()

# Initialize a figure
plt.figure(figsize=(12, 6))

# Plot the retention rate for the entire dataset, coloring by year
for year in [2015, 2016, 2017, 2018]:
    yearly_data = monthly_sales[monthly_sales["Month"].dt.year == year]
    plt.plot(yearly_data["Month"], yearly_data["Retention Rate"], marker="o",␣
 ↪linestyle="-", label=f"{year}")

# Set the title and labels
plt.title("Monthly Retention Rate of Sales (2015-2018)")
plt.xlabel("Month")
plt.ylabel("Retention Rate (%)")
plt.legend(title='Year')

# Set x-axis to show only the months
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b"))

plt.tight_layout()
```

```
plt.show()
```

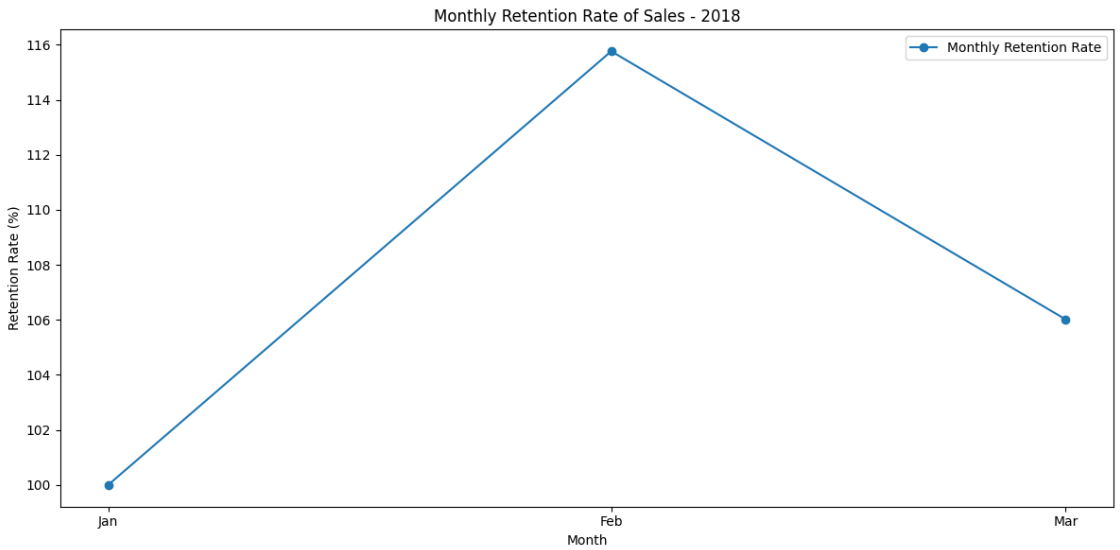
```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\3404341402.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')
```




[36]:
```
# Create a column for month and year
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')

# Filter the DataFrame to include only the data for the year 2018
df_2018 = filtered_df_regions[filtered_df_regions["Date"].dt.year == 2018]

# Group by month and calculate the total sales volume
monthly_sales_2018 = df_2018.groupby("Month")["Total Volume"].sum().
  ↪reset_index()

# Define sales cohorts in the first month as a baseline
first_month_sales_2018 = monthly_sales_2018["Total Volume"].iloc[0]

# Calculate sales retention rate: sales volume each month as a percentage of␣
  ↪the first month
monthly_sales_2018["Retention Rate"] = (monthly_sales_2018["Total Volume"] /␣
  ↪first_month_sales_2018) * 100
```

```
# Convert YearMonth to a date for the chart
monthly_sales_2018["Month"] = monthly_sales_2018["Month"].dt.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(monthly_sales_2018["Month"], monthly_sales_2018["Retention Rate"],␣
 ↪marker="o", linestyle="-", label="Monthly Retention Rate")

plt.title("Monthly Retention Rate of Sales - 2018")
plt.xlabel("Month")
plt.ylabel("Retention Rate (%)")
plt.legend()

# Set x-axis to show only the months of 2018
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b"))

#plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

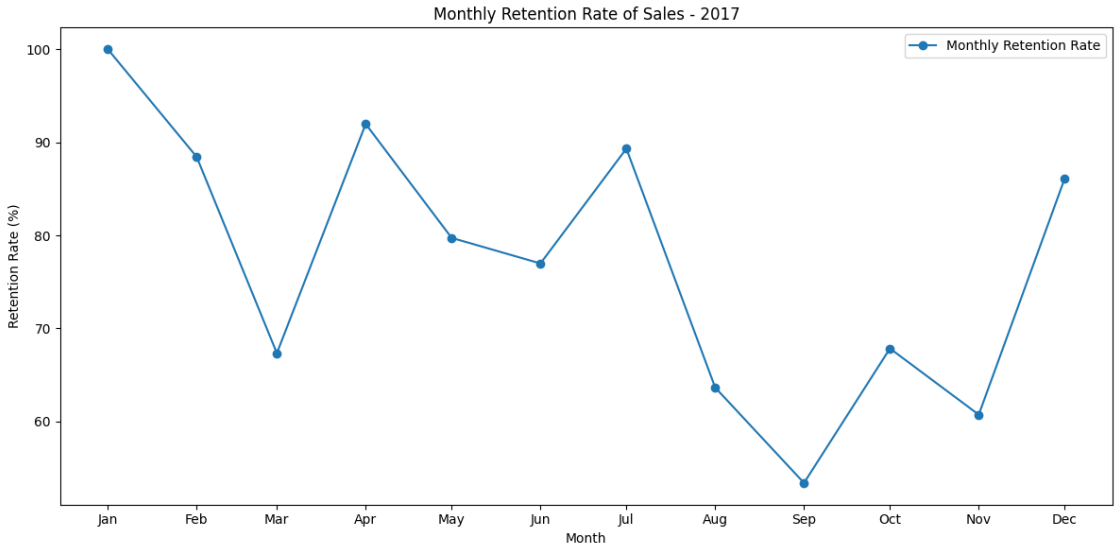
```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\3330760414.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')
```



[33]:
```
# Create a column for month and year
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')

# Filter the DataFrame to include only the data for the year 2017
df_2017 = filtered_df_regions[filtered_df_regions["Date"].dt.year == 2017]

# Group by month and calculate the total sales volume
monthly_sales_2017 = df_2017.groupby("Month")["Total Volume"].sum().
 ↪reset_index()

# Define sales cohorts in the first month as a baseline
first_month_sales_2017 = monthly_sales_2017["Total Volume"].iloc[0]

# Calculate sales retention rate: sales volume each month as a percentage of␣
 ↪the first month
monthly_sales_2017["Retention Rate"] = (monthly_sales_2017["Total Volume"] /␣
 ↪first_month_sales_2017) * 100

# Convert YearMonth to a date for the chart
monthly_sales_2017["Month"] = monthly_sales_2017["Month"].dt.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(monthly_sales_2017["Month"], monthly_sales_2017["Retention Rate"],␣
 ↪marker="o", linestyle="-", label="Monthly Retention Rate")

plt.title("Monthly Retention Rate of Sales - 2017")
plt.xlabel("Month")
plt.ylabel("Retention Rate (%)")
plt.legend()

# Set x-axis to show only the months of 2017
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b"))

#plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\2983205544.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

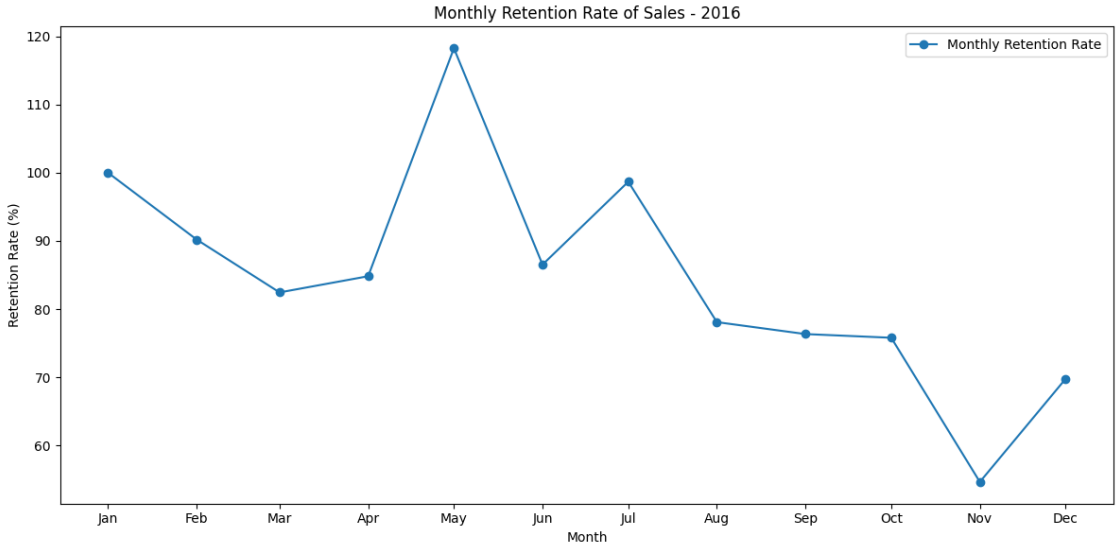
```
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')
```




[34]:
```
# Create a column for month and year
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')

# Filter the DataFrame to include only the data for the year 2016
df_2016 = filtered_df_regions[filtered_df_regions["Date"].dt.year == 2016]

# Group by month and calculate the total sales volume
monthly_sales_2016 = df_2016.groupby("Month")["Total Volume"].sum().
 ↪reset_index()

# Define sales cohorts in the first month as a baseline
first_month_sales_2016 = monthly_sales_2016["Total Volume"].iloc[0]

# Calculate sales retention rate: sales volume each month as a percentage of␣
 ↪the first month
monthly_sales_2016["Retention Rate"] = (monthly_sales_2016["Total Volume"] /␣
 ↪first_month_sales_2016) * 100

# Convert YearMonth to a date for the chart
monthly_sales_2016["Month"] = monthly_sales_2016["Month"].dt.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(monthly_sales_2016["Month"], monthly_sales_2016["Retention Rate"],␣
 ↪marker="o", linestyle="-", label="Monthly Retention Rate")

plt.title("Monthly Retention Rate of Sales - 2016")
```

```
plt.xlabel("Month")
plt.ylabel("Retention Rate (%)")
plt.legend()

# Set x-axis to show only the months of 2016
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b"))

#plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

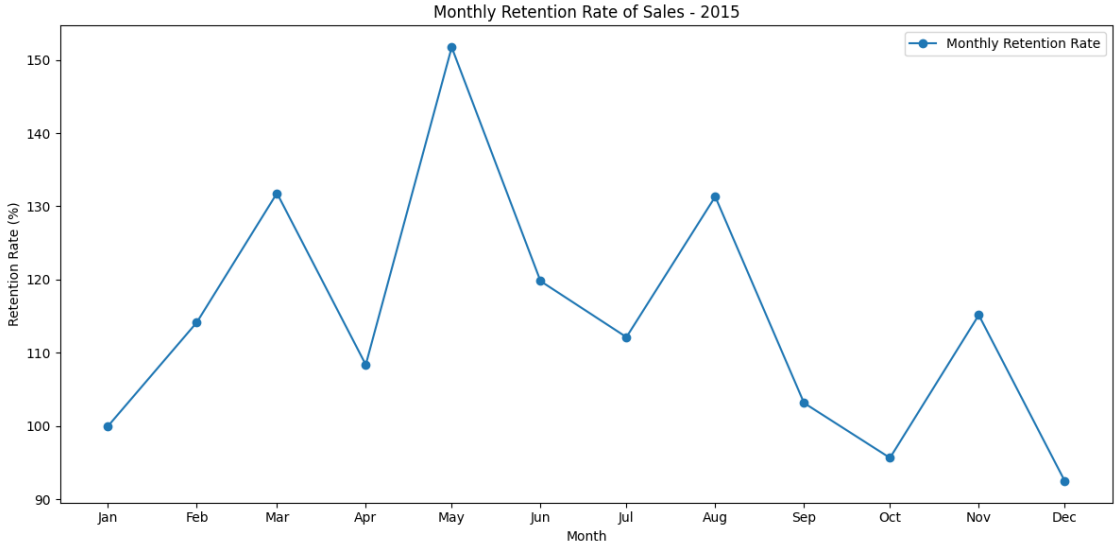
```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\1509764461.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')
```




[35]:
```
# Create a column for month and year
filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')

# Filter the DataFrame to include only the data for the year 2015
df_2015 = filtered_df_regions[filtered_df_regions["Date"].dt.year == 2015]

# Group by month and calculate the total sales volume
```

```
monthly_sales_2015 = df_2015.groupby("Month")["Total Volume"].sum().
 ↪reset_index()

# Define sales cohorts in the first month as a baseline
first_month_sales_2015 = monthly_sales_2015["Total Volume"].iloc[0]

# Calculate sales retention rate: sales volume each month as a percentage of␣
 ↪the first month
monthly_sales_2015["Retention Rate"] = (monthly_sales_2015["Total Volume"] /␣
 ↪first_month_sales_2015) * 100

# Convert YearMonth to a date for the chart
monthly_sales_2015["Month"] = monthly_sales_2015["Month"].dt.to_timestamp()

plt.figure(figsize=(12, 6))
plt.plot(monthly_sales_2015["Month"], monthly_sales_2015["Retention Rate"],␣
 ↪marker="o", linestyle="-", label="Monthly Retention Rate")

plt.title("Monthly Retention Rate of Sales - 2015")
plt.xlabel("Month")
plt.ylabel("Retention Rate (%)")
plt.legend()

# Set x-axis to show only the months of 2015
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b"))

#plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

```
C:\Users\egortega\AppData\Local\Temp\ipykernel_22764\1967854670.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df_regions['Month'] = filtered_df_regions['Date'].dt.to_period('M')
```


Monthly Retention Rate of Sales - 2015
Monthly Retention Rate
Retention Rate (%)
150
140
130
120
110
100
90
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month